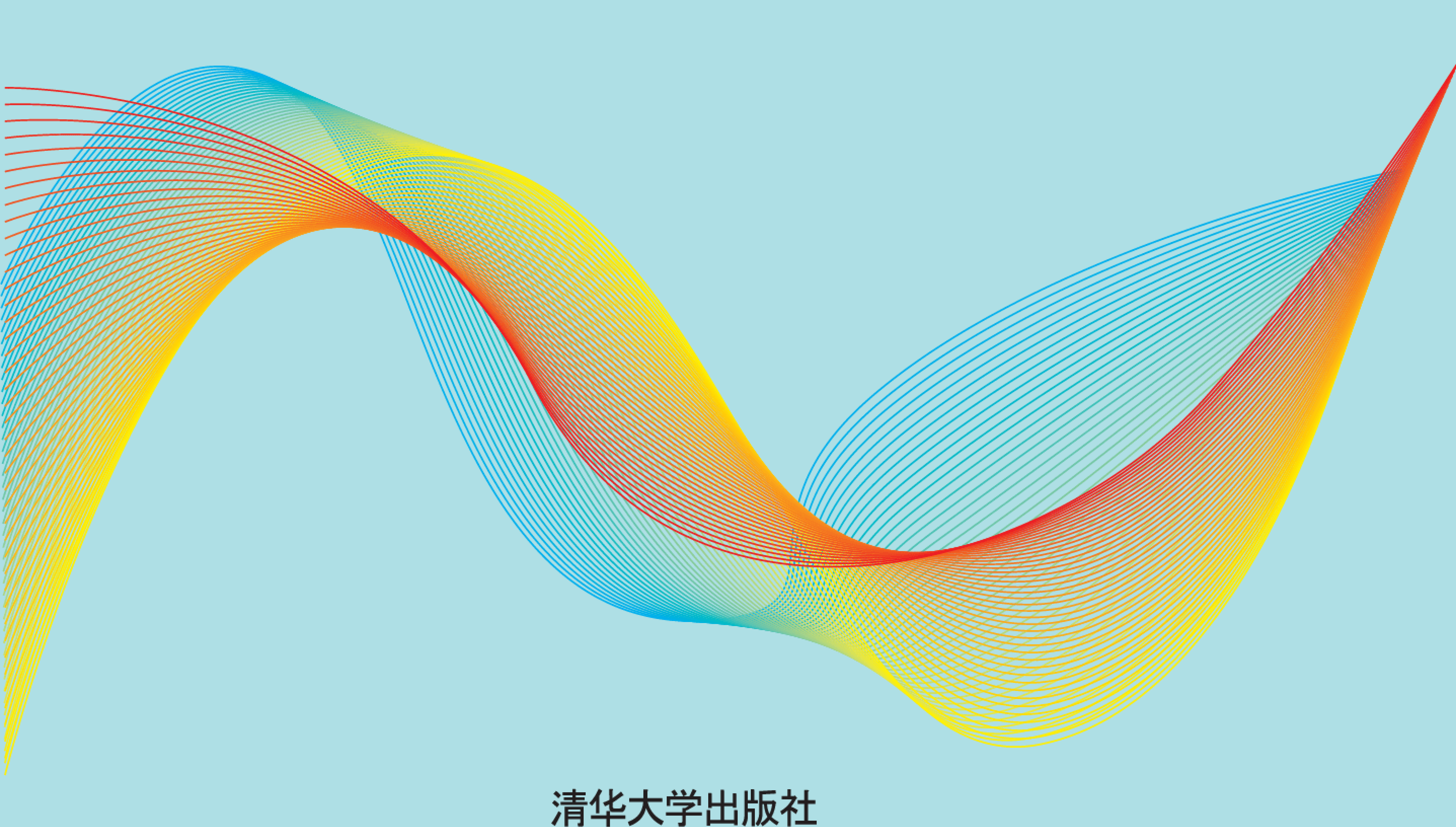
Ontology Model and Semantic Web Knowledge Discovery

本体建模与语义Web知识发现

阎红灿　著



清华大学出版社

河北省重点学科、国家自然科学基金资助

本体建模与语义 Web 知识发现

阎红灿 著

清华大学出版社
北 京

内 容 简 介

本书是作者多年来科研工作的梳理和升华，内容包括：XML文档管理和分类技术、知识资源描述语言和发布技术、本体建模和知识推理技术、基于知识库的知识发现关键技术和模型框架，同时给出了基于知识库的知识发现的典型应用。

本专著阐述明晰，内容新颖，力求深入浅出，可以用作高等院校有关专业的研究生和高年级本科生的信息检索、知识发现等课程教材，也可供从事数据挖掘、语义检索和知识管理的科技人员阅读参考。

图书在版编目(CIP)数据

本体建模与语义Web知识发现/阎红灿著. —北京：清华大学出版社，2015
ISBN 978-7-302-41627-2

Ⅰ. ①本…　Ⅱ. ①阎…　Ⅲ. ①知识工程　Ⅳ. ①TP182

中国版本图书馆CIP数据核字(2015)第228380号

责任编辑：付弘宇　柴文强
封面设计：
责任校对：梁　毅
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　**邮　　编**：100084
社 总 机：010-62770175　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：170mm×230mm　**印　　张**：18　**字　　数**：287千字
版　　次：2015年12月第1版　**印　　次**：2015年12月第1次印刷
印　　数：1～　000
定　　价：　.00元

产品编号：063882-01

前　　言

知识发现是从数据集中识别出有效的、新颖的、潜在有用的以及最终可理解的模式的非平凡过程。随着语义 Web 的发展，基于知识库的知识发现成为知识发现领域的一个研究目标。语义 Web 下经过 XML 标注的自然语言文本、资源描述框架 RDF 和本体建模为知识发现提供了有力工具，数据资源的描述和存储、索引，实现语义检索的知识推理，都是语义 Web 下基于知识库的知识发现的关键技术支撑。

专著全面而又系统地介绍了 XML 文档分类技术、RDF 知识资源描述语言和本体推理等知识发现的方法和技术，反映了当前本体建模与知识发现研究的最新成果和进展。全书共分 9 章。第 1 章是引言绪论，概述语义网络中智能检索的关键技术和基于知识库的知识发现的重要概念；第 2、3 章讨论 XML 数据库的存储策略，重点论述基于关系存储策略的索引、维护和更新技术，第 4 章论述 XML 网页的频繁模式挖掘和分类技术；第 5 章介绍一种基于 N 层向量空间模型的全文检索方法；第 6 章系统阐述知识表示和本体建模技术；第 7 章综述数据资源的描述和关联数据发布，阐述基于关联数据的知识发现模型；第 8 章讨论本体推理技术及粗逻辑在本体推理中的应用；第 9 章论述基于本体知识库的知识发现框架和关键技术，给出基于知识发现的案例推理应用模型。

书中内容新颖，高度总结和梳理了作者多年的科研成果，取材国内外最新资料，反映了当前该领域的研究水平。论述力求概念清晰，表达准确，算法丰富，突出理论联系实际，富有启发性。

为了丰富专著的内容，书中引用了参考文献中的学术资料，对这些研究成果和做出辛勤劳动的作者表示真诚感谢；专著出版得到了国家自然科学基金、河北省重点学科、华北理工大学以及清华大学出版社的大力支持，华

北理工大学理学院应用数学专业研究生王会芳、张奉等同学在本书的编写和校稿过程中做了一定工作，在此一并表示诚挚的谢意。

因为时间仓促，加上作者的学识浅陋，书中的不足与错误，敬请同行给予批评指正。

阎红灿

2015 年 6 月

目　　录

第 1 章　基于语义 Web 的智能检索和知识发现

Web 技术的出现使人类的生存空间得到极大扩展，并逐渐成为人们获取、传播和交换信息的重要途径。随着 Internet 的飞速发展和广泛应用，其缺陷也逐渐暴露出来，如搜索引擎智能程度低，搜索出来的结果往往不是用户真正需要的，检索结果是单一的网页等等。互联网的创始人 Tim Berners-Lee 于 2000 年 12 月 18 日在 XML2000 会议上正式提出语义 Web（Semantic Web）。语义 Web 的目标是使 Web 上的信息具有计算机可理解的语义，满足智能软件代理对万维网上异构和分布式信息的有效访问和搜索。语义 Web 不是另外一个 Web，它是现有 Web 的延伸，其中信息被赋予了良定义[1]。语义 Web 将在更加微小的信息之间建立直接的连接，例如一条街道的地址与一份地图等，用户可以将两个毫不相干的东西连接在一起，比如说银行报账单和日历。用户可以将银行报账单拖到日历上，也可以将日历拖到银行报账单上，这样就可以知道何时进行支付。语义 Web 将呈现给人们一个所有数据“无缝”式连接的网络。在语义 Web 技术破土而出之后，人们对 Facebook 和 MySpace 等社交网站的“痴迷”终将被“无所不连”的网络所取代。

1.1　语义 Web 下的信息检索

语义网的实现需要 3 大关键技术支持：XML、RDF 和 Ontology。Tim Berners-Lee 提出的语义网层次结构[2]如图 1-1 所示。

1）Unicode 和 URI 层：Unicode 和 URI 层是整个语义万维网的基础，

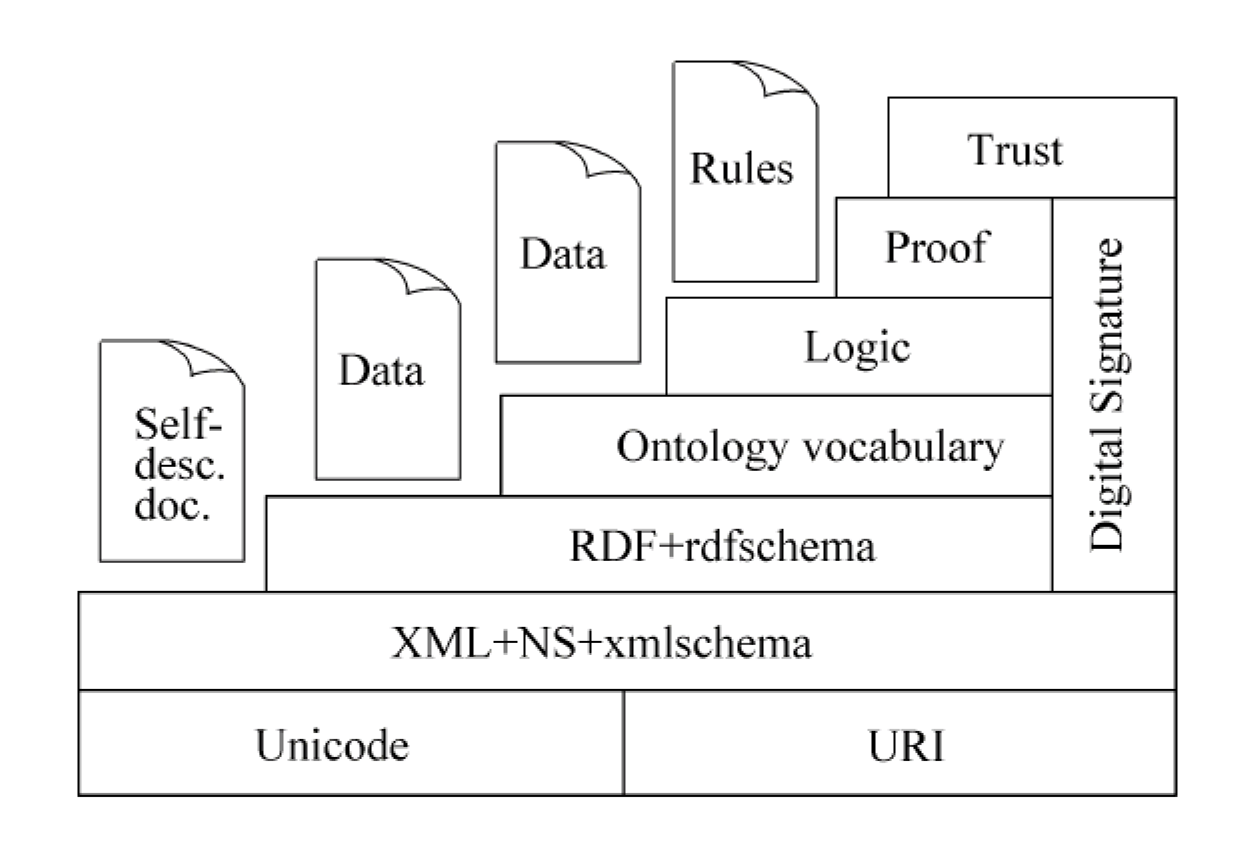


图 1-1 语义 Web 的层次模型

其中 Unicode(统一资源编码)处理资源的编码,保证使用的是国际通用字符集,实现网上信息的统一编码。URI 是 URL(Universal Resource Locator)的超集,URI 支持语义网上的对象和资源的精细标识,从而使精确信息检索成为可能。这一层是语义万维网的基石。

2) XML + Name Space + XML Schema 层

XML 和命名空间层用于表示数据的内容和结构,XML 层具有命名空间(Name Space)和 XML 模式(XML Schema)定义,通过 XML 标记语言可以将网上资源信息的结构、内容与数据的表现形式进行分离。

3) RDF + RDF Schema 层

RDF 和 RDFS 层用于描述万维网上的资源及其类型,为网上资源描述提供通用框架和实现数据集成的元数据解决方案。

4) 本体层

本体层用于描述各种资源之间的联系,采用 OWL 表示。本体(Ontology)揭示了资源以及资源之间复杂和丰富的语义信息,将信息结构和内容分离,对信息做完全形式化的描述,使 Web 信息具有计算机可理解的语义。

5) 逻辑层

逻辑层主要用于提供公理和推理规则,为智能推理提供基础。逻辑层可以进一步增强本体语言的表达能力,并允许创作特定领域和应用的描述性知识。

6）证明层

证明层设计实际的演绎过程以及利用 Web 语言表示证据，对证据进行验证等。证明注重提供认证机制，证明层执行逻辑层的规则，并结合信任层的应用机制来评判是否能够信任给定的证明。

7）信任层

信任层提供信任机制，保证用户 Agent 在 Web 上提供个性化服务，以及彼此之间安全可靠的交互，基于可信 Agent 和其他认证机构，通过使用数字签名和其他知识才能构建信任层。当 Agent 的操作时安全的，而且用户信任 Agent 的操作及其提供的服务时，语义 Web 才能充分发挥其价值。

在 Tim Berners-Lee 的语义网模型中，作为语法层的 XML 层，作为数据层的 RDF 层和作为语义层的 Ontology 层是语义 Web 的关键层。用于 Web 信息的语义，也是现在语义 Web 研究的热点所在。

语义 Web 下的信息检索，不再是一味的基于关键词的匹配，而是基于知识体系或概念网络的智能检索，即语义检索。比如用户查询“计算机”，与“电脑”相关的信息也能检索出来，甚至可以进一步缩小查询范围至“微机”、“服务器”或扩大查询至“信息技术”或查询相关的“电子技术”、“软件”、“计算机应用”等范畴。另外，智能检索还包括歧义信息和检索处理，如“苹果”，究竟是指水果还是电脑品牌，“华人”与“中华人民共和国”的区分，将通过歧义知识描述库、全文索引、用户检索上下文分析以及用户相关性反馈等技术结合处理，高效、准确地反馈给用户最需要的信息。

在信息检索分布化和网络化的趋势下，信息检索系统的开放性和集成性要求越来越高，需要能够检索和整合不同来源和结构的信息，这是异构信息检索技术发展的基点，包括支持各种格式化文件，如 TEXT、HTML、XML、RTF、MS Office、PDF、PS2/PS、MARC、ISO 2709 等处理和检索；支持多语种信息的检索；支持结构化数据、半结构化数据及非结构化数据的统一处理和关系数据库检索的无缝集成以及其他开放检索接口的集成等。所谓“全息检索”的概念就是支持一切格式和方式的检索，从实践来讲，发展到异构信息整合检索的层面，基于自然语言理解的人机交互以及多媒体信息

检索整合等方面尚有待进一步突破。

智能信息检索系统应具有如下的功能：

(1) 能理解自然语言，允许用自然语言提出各种询问；

(2) 具有推理能力，能根据存储的事实，演绎出所需的答案；

(3) 系统具有一定常识性知识，以补充学科范围的专业知识。系统根据这些常识，能演绎出更泛化的一些答案来。

语义 Web 为实现用户语义的智能检索提供技术平台和技术支撑。

1.2 知识和知识发现

知识[3]这一概念有三种比较有代表性的定义：

(1) Feigenbaum：知识是经过消减、塑造、解释、选择和转换的消息；

(2) Bernstein：知识是由特定领域的描述、关系和过程组成；

(3) Heyes-Roth：知识＝事实＋信念＋启发式。知识常常是模糊、不确定或不完全的，而且知识还处在不断地动态变化过程中。

1.2.1 知识描述

通常采用 Heyes-Roth 提出的知识的三维空间来描述任何知识，即知识的范围、知识的目的和知识的有效性。范围由具体到一般，目的从说明到指定，有效性由确定到不确定。知识的三维空间描述如图 1-2 所示。

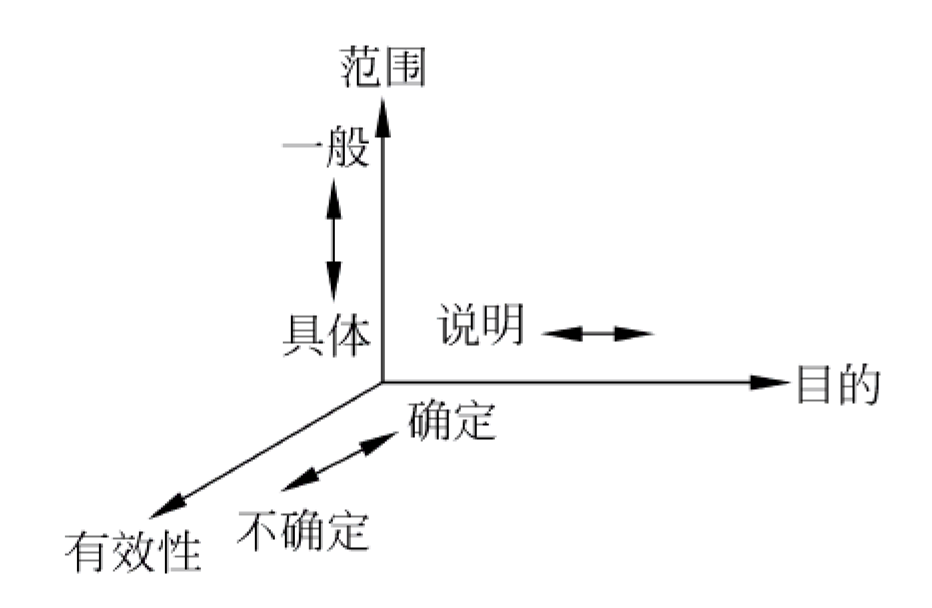


图 1-2 知识的三维空间描述

大量的数据经过加工后才会有价值，经过分析处理的数据形成了信息，信息的作用后有时间和范围的限制。为了使信息在较长的时间内有效，必须进行一系列的内部处理，这个过程叫综合，综合后的信息组成了知识。

从计算机科学的观点来看，知识是信息综合处理的结果。在综合过程中，信息传递相互比较，结合成有意义的链接。数据、信息和知识具有层次关系，它们的层次关系如图 1-3 所示。

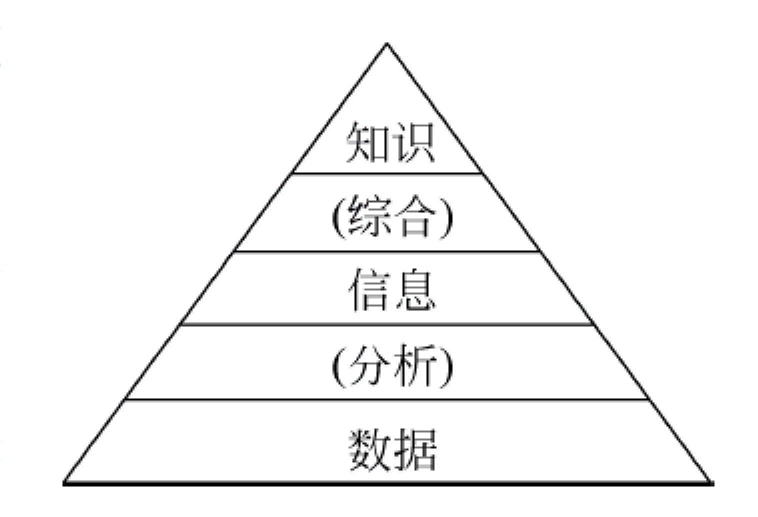


图 1-3　数据、信息和知识的层次关系

1.2.2　领域知识和知识库

领域知识主要应用在基于知识的专家系统和自然语言理解以及有关概念的约束的集合。知识工程对领域知识进行了三方面的描述：

(1) 领域知识是一个概念模型，这个概念模型包括概念和概念之间的关系；

(2) 领域知识是概念和概念之间的约束；

(3) 领域知识是陈述如何推导计算出新概念和新概念之间的关系的规则。

领域知识的两个基本概念：

(1) 领域特征概念：这是领域知识的概念化，是各种相关领域内的重要概念的语义描述；

(2) 领域特征属性：这是指某一领域内的概念所具有的特点，领域特征概念可以是词，也可以根据需要扩展成短语甚至词串。

知识库是针对某一领域问题求解的需要，采用某种知识表示方式在计算机存储器中存储、组织、管理和使用的互相联系的知识的集合。

领域知识是指在某一专门领域中重要问题或概念以及概念之间的相互关系的集合。领域知识库这一术语源于人工智能领域。在人工智能领域

中,领域知识主要应用于知识的专家系统和自然语言理解的系统。

1.2.3 基于知识库的知识发现

随着网络技术和通信技术的发展,数据、信息的产生和收集能力已经迅速提高。而更能为人们提供帮助的,潜在这些数据中的信息和知识却相对缺乏。这种数据爆炸而知识缺乏的情况激起人们对新技术和自动工具的需求,数据挖掘技术就是新技术之一。如何从长期积累的、大量的信息中找到对我们有用的知识已成为一个亟待解决的问题,于是知识发现应运而生。多研究者从不同的角度给出了有关知识发现的定义,目前较一致认同的描述性定义是 Fayyad 等人给出的:知识发现是从数据集中识别出有效的、新颖的、潜在有用的以及最终可理解的模式的非平凡过程。

传统的知识发现是指基于数据库的知识发现(KDD:Knowledge Discovery in Databases),是从大量的、不完整的、有噪声的、模糊的和随机的数据中,提取隐含在其中的、人们事先不知道的,但又是可信的、潜在的和有价值的信息和知识的过程。知识发现将信息变为知识,从数据矿山中找到蕴藏的知识金块,将为知识创新和知识经济的发展做出贡献。

知识发现与数据挖掘的关系密不可分。数据挖掘(Data Mining),就是从海量的数据中抽取出隐含的、未知的、具有潜在使用价值信息的过程[4-5]。由于数据挖掘是 KDD 过程中最为关键步骤,在实际应用中两个概念往往不加以区分。一般认为广义的数据挖掘又称数据库中的知识发现,简称知识发现(KDD);狭义的数据挖掘是一个利用各种分析工具在海量数据中发现模型和数据关系之间关系的过程,是知识发现过程的一个步骤,一个完整的知识发现过程如图 1-4 所示。从图中可见,数据挖掘是知识发现过程中一个发现模式的子过程,并且是最核心的过程。

完成从大型源数据中发现有价值知识的过程可以简单概括为:

首先从数据源中抽取出感兴趣的数据,并把它组织成适合挖掘的数据组织形式;然后调用相应的算法生成所需要的知识;最后对生成的知识模式进行评估,并把有价值的知识集成到企业的智能系统中。

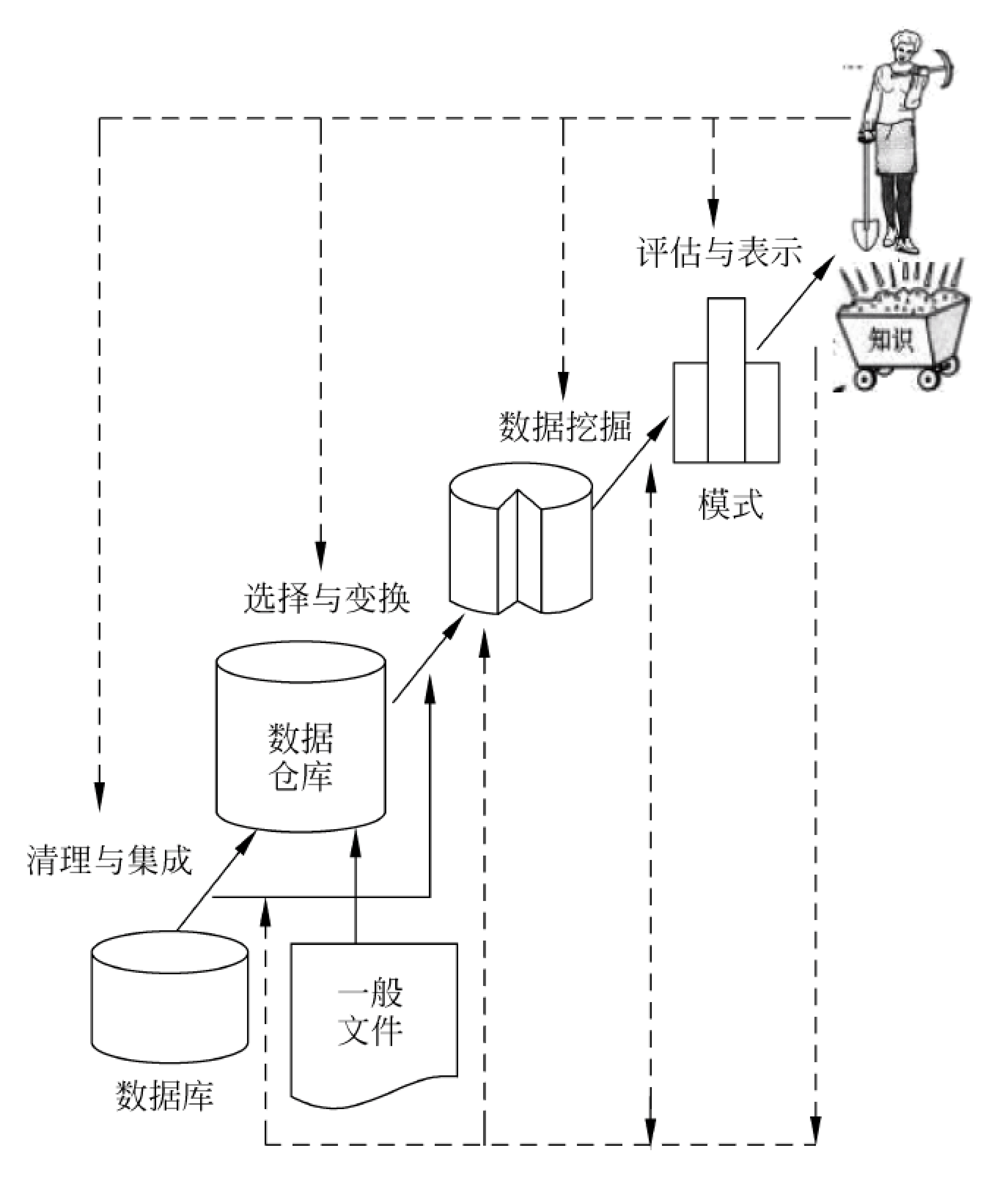


图 1-4　知识发现过程

作为一个 KDD 的工程而言，KDD 通常包含一系列复杂的挖掘步骤。Fayyad、Piatetsky-Shapiro 和 Smyth 在 1996 年合作发布的论文 *From Data Mining to knowledge discovery* 中总结出了 KDD 包含的 5 个最基本步骤：

（1）selection：在第一个步骤中我们首先要知道什么样的数据可以应用于我们的 KDD 工程中；

（2）pre-processing：当采集到数据后，下一步必须要做的事情是对数据进行预处理，尽量消除数据中存在的错误以及缺失信息；

（3）transformation：转换数据为数据挖掘工具所需的格式。这一步可以使得结果更加理想化；

（4）data mining：应用数据挖掘工具；

（5）interpretation/evaluation：了解以及评估数据挖掘结果。

传统的基于数据库的知识发现技术已经发展的比较成熟了，但在数据

向数据库进行存储的时候,并没有考虑到数据之间的语义信息,这就为知识发现带来了困难,在知识发现过程中不利于语义相关知识的发现。在所得到的大量信息中,包括有结构化的数据,半结构化的数据(文本信息等)和非结构化数据(图形、图像、视频、音频信息等)。而 KDD 技术只处理其中以结构化的数据类型。与变化不定的数据比较,知识是相对稳定的、是从较高的层次看数据、是数据的抽象表示,其表达的内涵要远远大于数据信息。因此,研究知识中内在的联系,将成为知识发现领域下一步的研究目标。于是基于知识库的知识发现应运而生。

在知识库建立过程中,引入本体概念,充分挖掘文档的语义信息,并以统一的格式描述知识即形成知识项存储于知识库中,为用户进行知识发现提供一种新的数据存储方式。以此为基础构建的知识发现系统,通过友好的用户界面为用户提供个性化知识发现服务,系统通过对用户的问题进行语义解析、推理机的推理以及语义映射等,从知识库中检索出用户满意的知识。

可见,基于知识库的知识发现的关键技术是本体建模、数据集成、语义解析和推理,而语义 Web 下的资源描述技术 XML、RDF 和本体推理技术为知识发现提供了有力工具。

1.3 XML 数据模式及应用

为了更好适应 Web 数据表示和交换的需要,国际标准化组织(International Standards Organization,ISO)于 1986 年发布了《标准通用标记语言》(Standard Generalized Markup Language,SGML)[6]。由于 SGML 过于繁琐,几乎没有应用能够支持这个标准,从而促使了 HTML(Hyper Text Markup Language)的出现。HTML 是为方便文档页面读者而设计的,其简单性极大地促进了 Web 的发展,使 Web 成为互联网上增长最快的领域。

随着互联网上信息量的进一步增长,各种媒体数据更广泛的应用,

HTML 的局限性也逐渐体现出来。近年来,电子商务、数字图书馆、远程教育等全新领域迅猛发展并逐渐成为互联网世界必不可少的组成部分,Web 文件的复杂化、多样化、高容量、高效率成为网络信息传输和处理技术发展追求的主要目标。与此同时,还有另一种需求变得愈发广泛而迫切,那便是同样的数据能否根据不同用户不同需求而以不同的效果、形式表达出来。虽然人们已付出很多努力,包括各式各样的修改扩充,如增加表格、框架、脚本语言等,但均未能从根本上改变其局限性。Web 提供商也发现普通的 HTML 已经无法提供大规模应用所需要的扩展性、结构性和数据检索功能。因此,他们每年都要对标准 HTML 定义的标签进行扩展以满足不断增长的 Web 需要,而不同 Web 厂商扩展的 HTML 经常互不兼容,造成一定的混乱。

由于 Web 信息通过 HTML 进行数据的表示和交换,管理 Web 信息的应用程序需要包含一个可以从 HTML 页面抽取信息结构与内容的“信息源包装程序”——Wrapper。这些包装程序非常脆弱,既要考虑不同 Web 厂商扩展的 HTML 特性,又要考虑应用的特性(如数据语义);并且其数据查询功能也十分脆弱。HTML 文档缺乏结构信息,一般的查询方法都是来自信息检索领域,只能提供基于关键字的查询,并且查询的结果通以整个文档为单位,从而造成网络带宽的浪费。

为解决 HTML 可扩展性差和 SGML 过于繁琐的缺陷,W3C(World Wide Web Consortium)组织于 1998 年 2 月发布了 XML 标准。XML 不仅在使用中得到各方肯定,而且在短短几年内迅速崛起,并得到 Microsoft、IBM 等各大公司的全力支持,成为一股不可遏抑的席卷全球的浪潮。该语言被描述为[1]:“XML 是 SGML 的子集,其目标是允许普通的 SGML 在 Web 上以目前 HTML(HyperText Markup Language)的方式被服务、接受和处理。XML 被设计成易于实现,且可在 SGML 和 HTML 之间互操作”。

1.3.1　XML 语言特点

XML 是一种元标记语言,根据需求用户可以定义自己的标记,相对于

HTML 而言，具有以下优点：

（1）自描述性

XML 文档通常包含一个文档类型声明，因而 XML 文档是自描述的。不仅人能读懂 XML 文档，计算机也能处理。XML 表示数据的方式真正做到了独立于应用系统，并且数据能够重用。XML 文档被看作是文档的数据库化和数据的文档化。

（2）先进特性

XML 继承了 SGML 的许多特性，首先是可扩展性。XML 允许使用者创建和使用他们自己的标记，而不是仅仅使用 HTML 的有限词汇表。这一点至关重要，企业可以用 XML 为电子商务和供应链集成等应用定义自己的标记语言，甚至特定行业的众多企业可以一起来定义该领域的特殊标记语言，作为该领域信息共享与数据交换的基础，XML 是源文档的最佳格式，便于信息交换。

（3）灵活性

HTML 很难进一步发展，就是因为它是格式、超文本和图形用户界面语义的混合，要同时发展这些混合在一起的功能是很困难的。而 XML 提供了一种结构化的数据表示方式，使得用户界面分离于结构化数据。所以，Web 用户所追求的许多先进功能在 XML 环境下更容易实现。

（4）支持文档内容验证

通过文档类型定义（Document Type Definition，DTD）或 XML 模式（XML Schema）方便地验证文档的有效性。

（5）支持高级搜索

在互联网上如果 Web 页是 XML 格式，则搜索可以附加数据的上下文信息，增加搜索效率。

有人认为 XML 是下一代 Web 语言，更或是 21 世纪的“世界语”，可见 XML 的巨大潜力和广阔的应用前景。

XML 是 SGML 语言的子集，但不同于 SGML，XML 去除了 SGML 中不便使用的各种特性。XML 文档中的标记不是标准定义的，而是用户根据

Web 数据表示和交换的需要自行定义的。由于利用 XML 语言可以创建具体领域的标记语言，因此业界一致认为 XML 是一种元语言。同时，XML 实现了数据和显示的分离，使 XML 更适合 Web 数据的表示和交换。表 1-1 给出了 XML 与 HTML 的对比，从中可以发现 XML 解决了 HTML 扩展性差的问题。XML 是一种简单、灵活的半结构化语言，它的出现为 Web 应用的发展勾画出美妙的前景。

表 1-1　XML 和 HTML 的对比

特点	HTML	XML
扩展性	不具扩展性	是一种元语言，可定义其他标记语言
侧重点	信息的表现形式	信息的结构
语法要求	不要求标记的嵌套、配对等，不要求标记之间有一定的顺序	严格要求嵌套、配对，并遵循一定的模式信息
可维护性	难以阅读、维护	结构清晰，便于阅读、维护
数据显示	内容描述与显示方式合为一体	内容描述与显示方式相分离
工具	有大量工具支持	工具尚不成熟

不同于传统的、结构化的关系数据，XML 数据是一种全新的半结构化数据，难以利用现有的关系数据库完成对 XML 数据的有效管理。因此，很多颇具影响的机构和大学开始研究如何存储和管理 Web 上急剧膨胀的 XML 数据，致力于 XML 数据访问性能的提高。从 2000 年至今，大量的研究工作致力于提高路径表达式的查询处理效率，并取得了显著的研究成果。

XML 文档本质上是保存信息的结构化载体。为了得到有效的 XML 文件，需要一种用来描述 XML 文档中信息结构的数据模型。不仅要建立 XML 文档中可以使用的 XML 词汇表，而且要定义 XML 文档中元素的顺序和元素的嵌套关系的内容模型，并建立文档数据的数据类型。

在 XML Schema 之前模式语言已经开发了：XDR（XML data reduced，XML 数据简化）、DCD（Document Content Description，文档内容说明）、SOX（Simple outline XML，简单 XML 概要）和 DDML（Document Definition Markup Language，文档定义标记语言）。W3C 于 1998 年开始制定 XML Schema 的第一个版本，2001 年 5 月正式推荐，有望成为主流的模式语言。

1.3.2 XML 模式的应用

XML 是一种全新的 Web 数据表示和交互标准，越来越多的 Web 数据通过 XML 格式进行存储和交互。从 1998 年出现至今，XML 技术受到了业界的广泛关注。Microsoft、IBM、Oracle 等参加了 XML 标准的制定，XML1.0[1]标准一出台，便开始了相应技术和商品的研制。Microsoft 的 Office、Windows 都将完全采用 XML 格式进行数据表示和交换；IE 浏览器更是早已实现了对 XML 的支持；IBM、Oracle 等公司也在各自的商品中提供了对 XML 应用的支持。目前 XML 应用比较成熟的领域主要有：

(1) 数据交换

在应用程序和公司之间作数据交换，首选用 XML，原因是 XML 使用元素和属性来描述数据。在数据传送过程中，XML 始终保留了诸如父子关系这样的数据结构。几个应用程序可以共享和解析同一个 XML 文件，不必使用传统的字符串解析或拆解过程。相反，普通文件不对每个数据段做描述(除了在头文件中)，也不保留数据关系结构。使用 XML 做数据交换可以使应用程序更具有弹性，因为可以用位置(与普通文件一样)或元素名(从数据库)来存取 XML 数据。

(2) Web 服务

Web 服务是最令人振奋的革命之一，它让使用不同系统和不同编程语言的人们能够相互交流和分享数据。其基础在于 Web 服务器可以用 XML 在系统之间交换数据。交换数据通常用 XML 标记，能使协议取得规范一致，比如在简单对象处理协议(Simple Object Access Protocol，SOAP)平台上。SOAP 可以在用不同编程语言构造的对象之间传递消息，这意味着一个 C# 对象能够与一个 Java 对象进行通信。这种通讯甚至可以发生在运行于不同操作系统上的对象之间。DCOM、CORBA 或 Java RMI 只能在紧密耦合的对象之间传递消息，SOAP 则可在松耦合对象之间传递消息。

(3) 内容管理

XML 只用元素和属性来描述数据，而不提供数据的显示方法。这样，

XML 就提供了一个优秀的方法来标记独立于平台和语言的内容。使用像 XSLT 这样的语言能够轻易地将 XML 文件转换成各种格式文件，比如 HTML、WML、PDF、flat file、EDI 等等。XML 具有的能够运行于不同系统平台之间和转换成不同格式目标文件的能力使得它成为内容管理应用系统中的优秀选择。

(4) Web 集成

现在越来越多的设备支持 XML，使得 Web 开发商可以在个人电子助理和浏览器之间用 XML 来传递数据。为什么将 XML 文本直接送进这样的设备去呢？这样做的目的是让用户更多地自己掌握数据显示方式，更能体验到实践的快乐。常规的客户/服务(C/S)方式为了获得数据排序或更换显示格式，必须向服务器发出申请；而 XML 则可以直接处理数据，不必经过向服务器申请查询-返回结果这样的双向“旅程”，同时在设备也不需要配制数据库，甚至还可以对设备上的 XML 文件进行修改并将结果返回给服务器。比如.INI 文件。虽然这样的文件格式已经使用多年并一直很好用，但是 XML 还是以更为优秀的方式为应用程序标记配制数据。使用.NET 里的类，如 XmlDocument 和 XmlTextReader，将配制数据标记为 XML 格式，能使其更具可读性，并能方便地集成到应用系统中去。使用 XML 配置文件的应用程序能够方便地处理所需数据，不用像其他应用那样要经过重新编译才能修改和维护应用系统。

1.4　知识表示和 OWL 本体语言

要使得知识能够在一定范围内共享、使用，就需要使用一种概括性强又能较为具体表示出知识之间关系的表示模型。知识模型的研究一直是知识工程领域的一个研究重点。知识表示是指把知识载体中的知识因子和知识关联表示出来，以便人们识别和理解知识。知识表示是知识组织的基础，因为任何知识组织的方法都必须建立在知识表示的基础上。目前，知识组织体系使用的知识表示方法很多，包含产生式规则、谓词逻辑式、语义网络、框

架式表示法、本体表示法等。基于知识本体的表示方法认为对知识本体的表示是一种可采用不同的方法来刻画自然世界的人为近似模型，它注重于知识表示的内容，而不是表现形式[7-8]。

本体描述语言，也称为标记语言、置标语言、构建语言或者是表示（标示）语言，是用特定的形式化语言对本体模型进行描述，使得机器和用户都能达到统一的理解。主要包括：XML、RDF/RDFS、OWL 等。如图 1-5 所示。

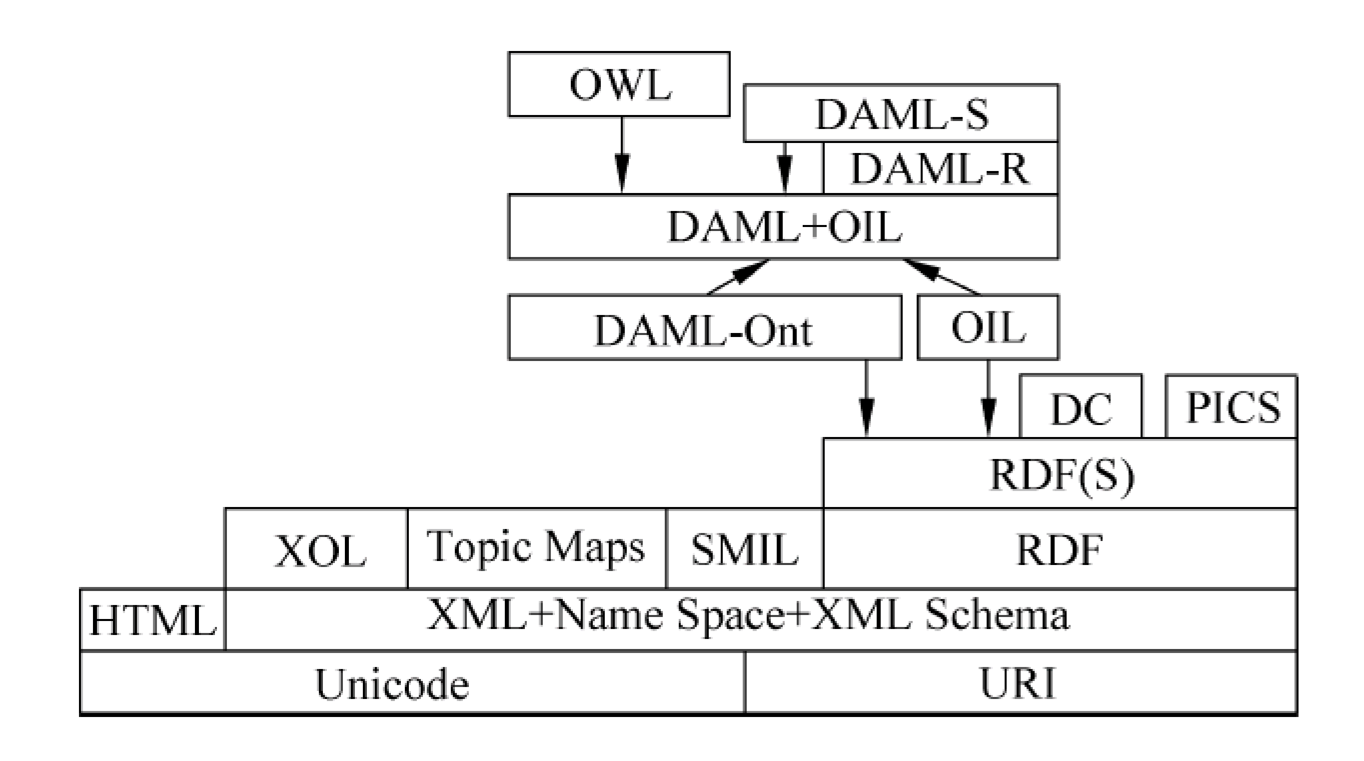


图 1-5　本体语言描述“堆”

XML 描述结构化文档的表层语法，对文档没有任何语义约束；XML Schema 定义 XML 文档的结构约束的语言；RDF 描述对象（资源）以及它们之间关系的数据模型，为数据模型提供简单的语义，这些数据模型能够用 XML 语法进行表达；RDF Schema 描述 RDF 资源的属性和类型的词汇表，提供了对这些属性和类型的普遍层次的语义；OWL 添加了更多用于描述属性和类型的词汇，例如类型的不相交性、基数（Cardinality）、等价性，属性更丰富的类型、属性特征以及枚举型（Enumerated Classes）等。

2000 年 XML 会议之后，W3C 为在网页上标注语义，积极推进了语义标注语言的研发。2001 年 2 月 W3C 正式推出 Semantic Web Activity。2004 年 2 月，在总结了 RDFS、DAML-ONT、DAML ＋ OIL 等本体描述语言的开发经验的基础上，W3C 将 Web 本体语言 OWL 作为推荐标准。OWL 提供了 3 种表达能力递增的子语言 OWL Lite、OWL DL 和 OWL Full，分别用于特定的用户群体。OWL Lite 用于提供给那些只需要一个分类层次和简单约束的用户，提供支持 OWL Lite 的工具应该比支持其他表达能力更强的

OWL 子语言更简单，并且从辞典（Thesauri）和分类系统（Taxonomy）转换到 OWL Lite 更为迅速，相比 OWL DL，OWL Lite 还具有更低的形式复杂度。OWL DL 用于支持强表达能力的同时需要保持计算的完备性（所有的结论都能够确保被计算出来）和可判定性（所有的计算都能在有限的时间内完成）的知识表示。OWL DL 包括了 OWL 语言的所有语言成分，但使用时必须符合一定的约束。OWL Full 支持最强的表达能力和完全自由的 RDF 语法的用户，但是 OWL Full 没有可计算性保证，同时，不太可能有推理软件能支持对 OWL Full 的所有成分的完全推理。OWL Full 可以看成是对 RDF 的扩展，而 OWL Lite 和 OWL DL 可以看成是对一个受限的 RDF 版本的扩展。所有的 OWL 文档（Lite、DL、Full）都是一个 RDF 文档；所有的 RDF 文档都是一个 OWL Full 文档，但只有一些 RDF 文档是一个合法的 OWL Lite 和 OWL DL 文档。实际应用中进行 OWL 子语言的选择时，选择 OWL Lite 还是 OWL DL 主要取决于用户在多大程度上需要 OWL DL 提供的表达能力更强的成分。选择 OWL DL 还是 OWL Full 则主要取决于用户在多大程度上需要 RDF Schema 的元建模（meta-modeling）机制（如定义关于类的类和为类赋予属性等）。

1.5　XML 为基于知识库的知识发现带来的希望和挑战

基于知识库的知识发现关键是知识库的建立和知识推理，知识库建立的过程是从信息中获取知识并以统一格式存储的过程。需要进行的工作包括领域本体的建立、文档的语义处理和知识项的建立。领域本体的建立主要是使用本体构建工具进行某一特定领域的本体的构建。文档的语义处理主要是对文档进行分词、语义标引等操作以抽取出文档中的语义信息，然后将这些语义信息连同本体与知识一起存储进知识库。

虽然语义网给我们展示了万维网的美好前景以及由此而带来的互联网的革命，但语义网的实现仍面临着巨大的挑战：

（1）内容的可获取性，即基于 Ontology 而构建的语义网网页目前还

很少；

（2）本体的开发和演化，包括用于所有领域的核心本体的开发、开发过程中的方法及技术支持、本体的演化及标注和版本控制问题；

（3）内容的可扩展性，即有了语义网的内容以后，如何以可扩展的方式来管理它，包括如何组织、存储和查找等；

（4）多语种支持；

（5）本体语言的标准化。

虽然与国外相比我国对语义网的研究相对落后，但从 1999 年至今发表的论文来看，①论文数量逐年递增。2002 年发表相关论文 22 篇，分别是 2000 年（6 篇）和 2001 年（4 篇）年论文数量的 3.7 倍和 5.5 倍，2003 年发表论文 38 篇，是 2002 年的 1.7 倍，这说明随着时间的推移，对语义网的研究已经引起了我国学者的高度重视；②研究内容越来越广泛而深入，大致可分为三个层次：

第一层次，即对语义网及其关键技术的描述与介绍，主要包括语义网的含义、体系结构、关键技术（RDF、Ontology）等；

第二层次，是关于语义网及其关键技术对相关学科或研究领域的影响与启示，包括信息管理、信息检索、知识库系统、数字图书馆、数据挖掘、电子商务、机器翻译、智能代理、需求分析、元数据描述与交换、网络信息资源和知识的表达等；

第三层次，则是针对语义网及其关键技术所做的具体试验与应用，包括 RDF 的应用与存储、基于 RDF/XML 的搜索引擎的设计与实现、语义网的试探性实现、Ontology 的构建、基于 Ontology 的查询系统设计、Ontology 在图书服务网络、知识图书馆和数字图书馆中的应用、Ontology 与主题词表相结合实现对元数据的查询等。

图 1-1 中 xmlschema 就是 XML 模式，rdfschema（RDF 模式）就是“资源描述框架模式”。其进一步的描述如图 1-6 所示。

RDF 和 RDF 模式中主要的类别、特性和约束如下：

（1）核心类别：包括 rdfs：Resources，rdfs：property 和 rdfs：Class。所

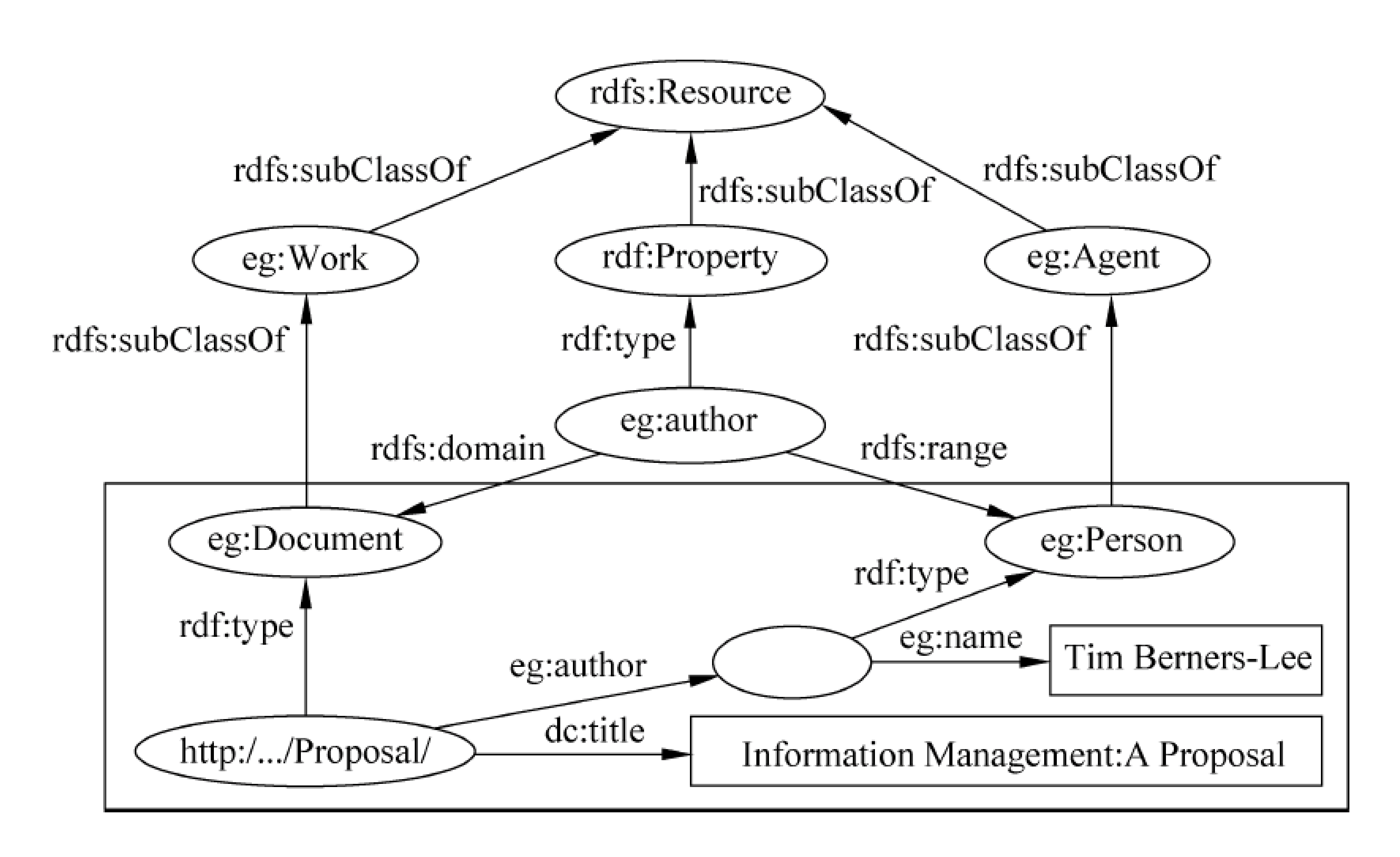


图 1-6　RDF 模式

有用 RDF 表达式描述的事物都被看成是 rdfs:Resources 的实例，rdfs:property 用来刻画 rdfs:Resources 实例的所有特性的类别，rdfs:Class 用来定义 RDF 模式中的概念。

（2）核心特性：包括 rdf:type，rdfs:subClassOf 和 rdfs:subPropertyOf。rdf:type 建立资源和类别之间的实例（instance-of）关系模型，rdfs:subClassOf 建立类别之间的包容层次模型，rdfs:subPropertyOf 建立特性之间的包容层次模型。

（3）核心约束：包括 rdfs:ConstraintResources，rdfs:ConstraintProperty，rdfs:Range 和 rdfs:domain。rdfs:ConstraintResources 定义了所有约束的类别，rdfs:ConstraintProperty 是 rdfs:ConstraintResources 和 rdf:Property 的子集，它包括所有用来定义约束的特性。

因此，RDF 和 RDF 模式是开放的元数据框架，它定义了一种描述计算机可理解的数据语义的数据模型。

语义互联网体系结构中的 Ontology Vocabulary 就是“本体词汇”，Logic 就是“逻辑”，它使用“描述逻辑标记语言”（Description Logic Markup Language）来进行描述，Proof 就是“验证”，Trust 就是“信任”，“验证”和“信任”是网络传输不可或缺的重要因素，用户与用户之间，用户与网络服务器

之间要进行验证,要保持诚信,必须要有“验证”和“信任”,用户与网络服务器之间才可以在网络上传输信息。

语义互联网体系结构左侧的 Self-desc. doc 是 Self-description document 的缩写,就是“自描述文档”,Data 是“数据”,Rule 是“规则”。

特别值得注意的是,在这个语义互联网的体系结构中,“RDF＋rdf 模式”的上面是“本体词汇”,“本体词汇”的上面是“验证”和“信任”,“本体词汇”处于语义互联网的关键层,它属于“自描述文档”,用于表示语义互联网各种信息的概念和语义。由此可见,“本体词汇”在语义互联网的整个体系结构中起着承上启下的联系作用,处于举足轻重的地位。

采用“本体词汇”来描述语义互联网中各种资源之间的联系,可以克服目前万维网上的信息格式的异构性、信息语义的多重性以及信息关系的匮乏和非统一性等严重问题。

其实,语义互联网的体系结构中的“资源描述框架模式”(rdfschema)就可以定义类别、子类和超类,并且可以定义特性和子特性以及对于特性的约束,例如,领域(domain)、范围(range)等等。但是,“资源描述框架模式”对自然语言和特定应用领域的词汇的描述能力较弱,需要根据自然语言词汇的特性和特定领域的术语词汇进一步扩展,因此,蒂姆·伯讷斯-李在“RDF＋rdfschema”上面再加上一个扩展层,这样的扩展层就是“本体词汇”。

2006 年 5 月,蒂姆·伯讷斯-李又宣布,经过十年的努力,W3C 已发布推荐标准 80 余份,语义互联网已经具备了为达到成功的目标所需要的所有标准和技术,包括作为数据语言的 RDF、本体语言、查询和规则语言。2006 年 4 月,万维网联盟中国办事处成立并召开了 WWW 技术研讨会。

蒂姆·伯讷斯-李在 2006 年又公布了语义互联网的新的体系结构[9],如图 1-7 所示。

在这个新的体系结构中,“逻辑”层变成了“统一逻辑”层(Unifying Logic),不再局限于使用特定的“描述逻辑标记语言”。

在这个体系结构中,Ontology Vocabulary 变成了 Ontology OWL,其中,OWL(Ontology Web Language)可以翻译成“本体网络语言”,是 W3C

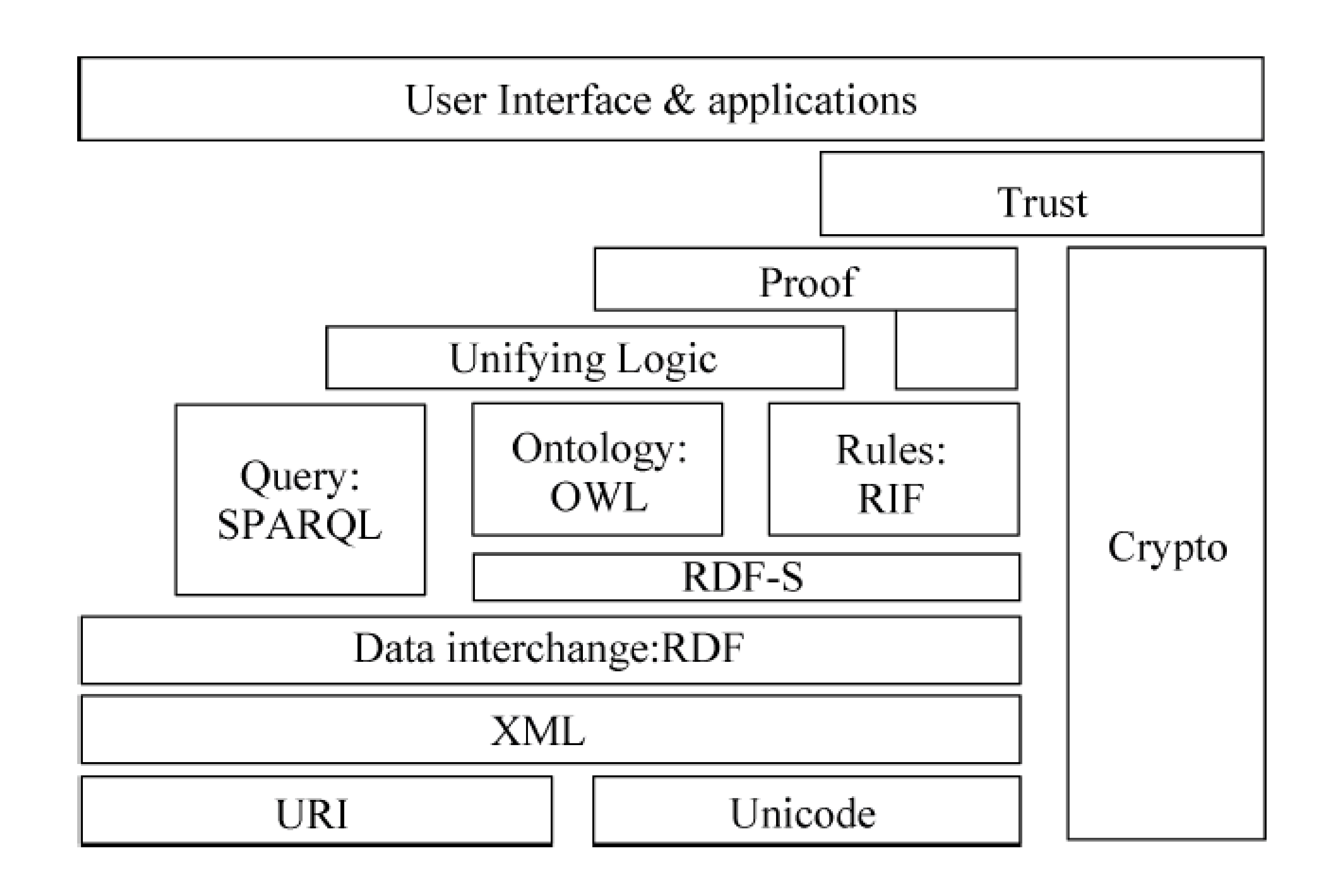


图 1-7　语义 Web 的新体系结构(2006)

开发的一种描述本体的网络语言,用于对本体进行语义描述。W3C 的设计人员针对各类特征的需求制定了三种相应的 OWL 的子语言,即 OWL Lite、OWL DL 和 OWL Full,而且各子语言的表达能力递增。

OWL Lite 是表达能力最弱的子语言。它是 OWL DL 的一个子集,但是通过降低 OWL DL 中的公理约束,保证了迅速高效的推理。它支持基数约束,但基数值只能为 0 或 1。因为 OWL Lite 表达能力较弱,为其开发支持工具要比其他两个子语言容易一些。OWL Lite 用于提供给那些仅需要一个分类层次和简单约束的用户。

OWL DL(Description Logic,描述逻辑)将可判定推理能力和较强表达能力作为首要目标,而忽略了对 RDFS 的兼容性。OWL DL 包括了 OWL 语言的所有语言成分,但使用时必须符合一定的约束,受到一定的限制。OWL DL 提供了描述逻辑的推理功能,描述逻辑是 OWL 的形式化基础。

OWL Full 包含 OWL 的全部语言成分并取消了 OWL DL 中的限制,它将 RDFS 扩展为一个完备的本体语言,支持那些不需要可计算性保证(no computational guarantees)但需要最强表达能力和完全自由的 RDFS 用户。在 OWL Full 中,一个类可以看成是个体的集合,也可以看成是一个个体。由于 OWL Full 取消了基数限制中对可传递性质的约束,因此不能保证可判

定推理。

“本体网络语言”直接地处于 Unifying Logic(统一逻辑)和 RDF-S(资源描述框架模式)之间,上承“统一逻辑”,下启“资源描述框架模式”,其承上启下的作用更加明显。由此可见,在 2006 年新公布的语义互联网体系结构中,“知识本体”的重要性更加突出了。

从 2001 年和 2006 年的语义互联网的两个体系结构可以看出,经过 XML 标注的自然语言文本是语义互联网的基础,Web 主要是由语言文字组成的(此外还有语音、音乐和图像),经过 XML 标注了句法、语义等信息之后的“文本文档”(text file),再经过资源描述框架 RDF 的处理、本体知识处理、规则处理、统一逻辑处理之后,就成为了“智能文档”(intelligent file)。

智能文档是“知道”自己内容的文档,其目的是让自动化程序“知道可以用它来做什么”。这些自动化程序叫做“智能代理”(Agent),智能代理是实现语义互联网服务(Semantic Web Service,SWS)的重要构件。语义互联网用知识本体(ontology)来表示概念以及概念之间的关系,所以文档的语义信息标注实际上是一种建立在知识本体基础之上的标注。

这些技术为语义 Web 下基于知识库的知识发现奠定了坚实的技术基础,也给研究者提出了新的挑战和机遇。

参考文献

[1] World Wide Web Consortium. Extensible Markup Language (XML) 1. 0 (third edition). W3C Recommendation. 4 February 2004. http://www. w3. qrg/TR/REC-xml/
[2] Barners Lee T, Hendler J. Lassila O. The semantic web. New York: Scientific American, 2001
[3] 马晓丹,邓晓楠,彭文娟等. 基于领域本体的知识库架构和实现[J]. 河北联合大学学报,2012(10): 42-47
[4] 纪希禹. 数据挖掘技术应用实例[M]. 北京: 机械工业出版社,2009
[5] 姚家奕. 数据仓库与数据挖掘技术原理及应用[M]. 北京: 电子工业出版社,2009. 8
[6] International Organization for Standardization: Information processing, Text and office systems, Standard Generalized Markup Language. Available at: http://www.

iso. org/iso/en/CataloguesDetailPage
[7]　董坚峰，胡凤. 基于 OWL 本体的知识表示研究[J]. 情报理论与实践. 2010，33(9)：89-92
[8]　王珏，袁小红，石纯一. 关于知识表示的讨论[J]. 计算机学报，1995，18(3)：212-224
[9]　冯志伟. 语义互联网与知识本体[EB/OL]，http://blog. sina. com. cn/s/blog_72d083c701017row. html. 2011. 6

第2章　XML与数据库

众所周知，当大量数据需要处理分析时，最好的方法是把这些数据放到数据库中，所以几乎所有大型的商业应用系统都和数据库相关联。XML本身是不是数据库，从严格的意义上来说，XML仅仅意味着XML文档。因为尽管一个XML文档包含数据，但是如果不通过其他的软件进行数据处理，它本身只不过是一个文本文件。所以XML本身不能和数据库挂上钩，但是加上一些其他的辅助工具，就把整个XML看成是一个数据库系统，XML文本本身可以看成是数据库中的数据区，DTD或者Schema可以看成是数据库模式，XQL(XML Query Language，XML查询语言)可以看成是数据库查询语言，SAX(Simple API for XML)或DOM(Document Object Model，文档对象模型)可以看成是数据库处理工具。当然它还是缺少数据库所必须的一些东西，比如有效的存储组织、索引结构、安全性、事务处理、数据完整性、触发器、多用户处理机制等等。

但是为什么要把XML和数据库相联系呢？举个例子来说明这个问题，比如有一个电子商务的应用程序需要使用XML来进行数据传输，最关心的是数据本身应该具有的结构，并不关心它在文档中实际的存储结构。如果应用程序很简单，基本的文件系统将满足需求，但如果应用本身很复杂，就需要一个完整的开发应用环境来支持XML。从另一个方面来说，假设有一个Web站点，它的内容是由一系列XML文档构成的，不仅要管理这个站点，同时需要提供给用户一个搜索该站点内容的机制，而这些都需要借助数据库来实现。选择一个数据库的最重要的因素是判断是否需要数据库来存储数据或者是文档。如果想要存储数据，需要一个关系数据库或者是对象数据库来存储实际的数据，同时需要中间件在数据库和XML文档之间建立

桥梁关系。从另一方面来说，如果想要存储文档，就需要一个内容管理系统，通过它进行文档的存储。实际上，XML 文档可以分为两大类：以数据为中心的文档和以文档为中心的文档。

（1）以数据为中心的文档：以数据为中心的文档有非常规则的结构，比如关于销售订单或者是饭店菜单的 XML 文档，通常是为机器设计的，主要方便机器进行处理，用作数据的传输载体。一般来说，任何 Web 站点都可以动态的构建 HTML 文档，其步骤如下：根据用户的查询请求找到相关的面向数据的 XML 文档，然后通过 XSL(eXtensible Stylesheet Language，可扩展样式表语言)对 XML 文档进行转化，让基于 HTML 的浏览器方便浏览结果。

（2）以文档为中心的文档：以文档为中心的文档具有不规则的结构，而且数据的粒度也比较大。如书本、电子邮件、广告等。以文档为中心的文档主要是为人消费而设计的，通常是以 XML 手工写成，或从其他格式(如 RTF、PDF、SGML)转换到 XML。

为了存储或提取数据，可以使用数据库和中间件，或者可以使用 XML 服务器，或者是基于 XML 的 Web 服务器；为了存储文档，需要一个内容管理系统或者是可持久化的 DOM 实现。如何有效地存储和查询 XML 数据是当前研究的一个热点。在存储和查询 XML 数据这一领域，主要有两种方法：一种是为 XML 数据量身定做的数据库即纯 XML 数据库。纯 XML 数据库充分考虑到 XML 数据的特点，以一种自然的方式来处理 XML 数据，能够从各方面很好地支持 XML 的存储和查询，并且能够取得较好的效果，但是纯 XML 数据库要走向成熟还有很长的路。第二种是在已有的关系数据库系统或面向对象数据库系统的基础上扩充相应的功能，使其能够胜任 XML 数据的处理，这种数据库又称为 XML 使能数据库。目前，XML 使能数据库的研究主要是基于关系数据库[1]，可以充分利用已有的非常成熟的关系数据库技术，集成现有的大量存储在关系数据库中的商用数据，但这种处理方法不能利用 XML 数据自身的特点，如结构化、自描述性等特征，使得在处理 XML 数据的时候要经过多级复杂的转换，如存储 XML 数据时要将其转换为关系表或对象，在查询的时候要将 XML 查询语言转换为 SQL，查

询结果还要转换为 XML 文档等,多级转换必将带来效率的降低。

2.1 XML 数据库分类及存储

目前 XML 数据库有下面三种类型[2]:

(1) 纯 XML 数据库(Native XML Database,NXD)。其特点是以自然的方式处理 XML 数据,以 XML 文档作为基本的逻辑存储单位。针对 XML 数据存储和查询特点专门设计适用的数据模型和方法。

(2) 使能 XML 数据库(XML Enabled Database,XEDB)。其特点是在原有的数据库系统上扩充对 XML 数据的处理功能,使之能适应 XML 数据存储和查询的需要。一般的做法是在数据库系统之上增加 XML 映射层,可以由数据库供应商提供,也可以由第三方厂商提供。映射层管理 XML 数据的存储和检索,但原始的 XML 元数据和结构可能会丢失,而且数据检索的结果不保证是原始的 XML 形式。XEDB 的基本存储单位与具体的实现紧密相关。

(3) 混合 XML 数据库(Hybrid XML Database,HXD)。根据应用的需求,可以视其为 XEDB 或 NXD 的数据库,比较典型的例子是 Ozone[3]。

因为底层的存储表达对上层的查询处理和优化有着重要的性能影响,所以如何以最好的方式存储 XML 文档已经成为一个重要问题。根据已有的文献[4~8],XML 数据库的存储策略主要有以下四种:利用文件系统的平面文件、利用成熟的 RDBMS(Relational Database Management System,关系数据库管理系统)、利用对象管理器或 OODBMS(Object-Oriented Database Management System,面向对象数据库管理系统)、采用全新的 Native XML 数据库管理系统。

2.2 纯 XML 数据库的存储结构和检索技术

关于纯 XML 数据库,R. Bourret 给出了一个定义,即只有满足以下 3 个条件的 XML 数据库才能称之为纯 XML 数据库[9]:

（1）为 XML 文档定义了一个（逻辑）模型，XML 数据的存储和查询都基于这个模型。这个模型至少要包含元素、属性以及 PCDATA 等，并保持文档顺序。

（2）将 XML 文档作为（逻辑）存储的基本单位，正如关系数据库将行（元组）作为存储的基本的单位一样。

（3）不要求只能使用某一特定的底层物理模型或某种专有的存储格式。

从定义中可以看出，纯 XML 数据库的核心在于其模式，即它的逻辑模式必须是某种特殊的模型，而不能是关系的或面向对象的。换言之，纯 XML 数据库与 XML 使能数据库的本质区别是其（逻辑）模式不同，而不是其底层的存储方式不同。

2.2.1　纯 XML 数据库的存储结构

存储结构是索引、查询处理与优化以及事务和并发控制的基础。简略地说，XML 存储管理的任务是：一方面能够有效地存储 XML 数据，另一方面要能够有效地支持查询和更新。

1. 存储方案

纯 XML 数据库在物理上存储 XML 数据主要有三种方案：

（1）平坦的流方式（flat streams）或者称之为基于文本的方式（text-based），即将 XML 数据转换成字节流，然后将其存储在文件系统的文本文件中，或存储为数据库的 BLOB 字段中，然后在这些文件或字段上建立一些索引。这种方法的优点是当存储或检索整个文档或连续的文档片断时速度很快，并且能够精确地再现原来的 XML 文档，但是当重组整个文档或者提取文档的结构时效率很低，因为它只有通过对整个文档的解析才能实现。

（2）建立模型或称之为基于模型的方式（model-based），即按照某种（物理）模型存储 XML 文档。根据模型的不同，又分为两种方案：一种是采用关系的或面向对象的数据库作为数据的储存库，另一种是为 XML 数据库设计专有的存储方案。前一种方案能够利用现有的关系或对象数据库技术

(如并发控制的技术),并且在重组文档片断或不同文档时比较快,但是,在逻辑层和物理层的数据需要经过转换,因而会降低处理效率。后者如 DOM 或它的变体,比如 Infonyte DB[10] 采用的 PDOM 方式就是先将文档转换为 DOM 结构,然后将其映射到一些特殊文件中。这种方案能够以一种比较自然的方式来存储 XML 数据,避免转换,但由于采用全新的存储方案,其他方面的技术不如前者成熟。

(3) 综合前面两种方式的特点,称为混合型(mixed),细分为冗余型(redundant)和杂交型(bybrid)。冗余型是指每份数据保持两份副本,一份是基于文本方式存储,一份基于模型存储。这样可以同时利用两种方式的优点,但是如果两种方式处于不一致状态,则更新效率很低。在杂交型存储方案中规定一个数据单元,粒度大于这个数据单元的部分以模式方式存储,而粒度较小的部分则不再细分,直接平面存储,Natix[11] 就是采用这种存储方式。

在实际的纯 XML 数据库中用得比较多的是基于模型的方式和杂交方式。文献[10]给出了一系列纯 XML 数据库产品列表,有些产品的 XML 存储结构是从文档处理的角度出发而不是从数据库的角度出发考虑的,如它们的结构是过程性的、一次一元组的,比较适合处理小文档。也有一些系统构建在传统数据库之上,具有多级转换和处理大数据效率较低的缺点。

2. 记录的物理存储和存储粒度

尽管各种纯 XML 数据库基于的模型不一样,但基本原理都是抽象成一个由各种结点组成的树状模型,树中常见的结点有元素结点、属性结点、文本结点等。Infonyte DB 完全是基于 DOM 模型的。Lore[12] 中的数据模型是称为 OEM(object exchange model)的一种有向边标记图(labeled directed graph),这个图中的各个对象基本上就对应了各个元素结点,而属性和文本结点则没有作为单独的对象表示出来。在 Timer[13] 中采用了一种 DOM 树的变体,在它的树模型中,每个元素都对应着一个结点,子元素对应着子结点,元素结点的内容也对应着一个结点,而元素结点的所有属性聚集为一个

孩子结点；如果元素是混合类型，则它的对应部分相应的抽取成为一个孩子结点。

在物理存储中，存取的最小单位是记录，每个记录都有自己的 ID，因此在决定存储方案时就要考虑三个关键问题：一是记录与结点的对应关系，即记录的粒度；二是记录的顺序；三是记录的内部表示，即当一个记录对应于多个结点时，结点在记录内部如何表示。

记录的粒度主要有三种：

（1）结点级：每个结点对应一个记录。结点在不同的系统中含义不同，有的系统中结点泛指元素结点（element node）、属性结点（attribute node）、文本结点（text node）和混合结点（mixed node），有的系统则仅把元素结点和该元素结点的属性结点和文本结点看作一个结点。

（2）子树级：一个子树对应一个记录。这种方法的关键在于如何划分子树，可根据物理块大小划分，也可根据逻辑意义划分。

（3）文档级：将整个文档作为一个记录。

这几种粒度的记录组织方式各有特点，记录的粒度越小，记录数目就越多，用以表示记录之间联系的指针（物理的或逻辑的）就越多，冗余空间需要的也越多，从而使得记录的存储效率降低，但是粒度小的记录在重组文档时可以避免不必要的转换和解析；记录的粒度越大，记录表示的结点就越多，因此在构造记录、向记录中插入结点和分裂记录时就越麻烦。

在几种常见的纯 XML 数据库中，Lore 和 Timer 是记录结点级的例子，Natix[14] 和 OrientX[15-17] 是记录子树级的例子，OrientX 是中国人民大学孟小峰教授等研究开发的一个纯 XML 数据库系统，Xindice[18] 是文档级记录的例子。

3. 记录的组织顺序

对于结点级或子树级记录来说，还有一个记录的组织顺序问题，其实文档级记录也有记录的顺序问题，因为实际的数据库要存储多个文档。记录的存储顺序一般有以下几种：按深度优先存储；按广度优先存储；按同类记

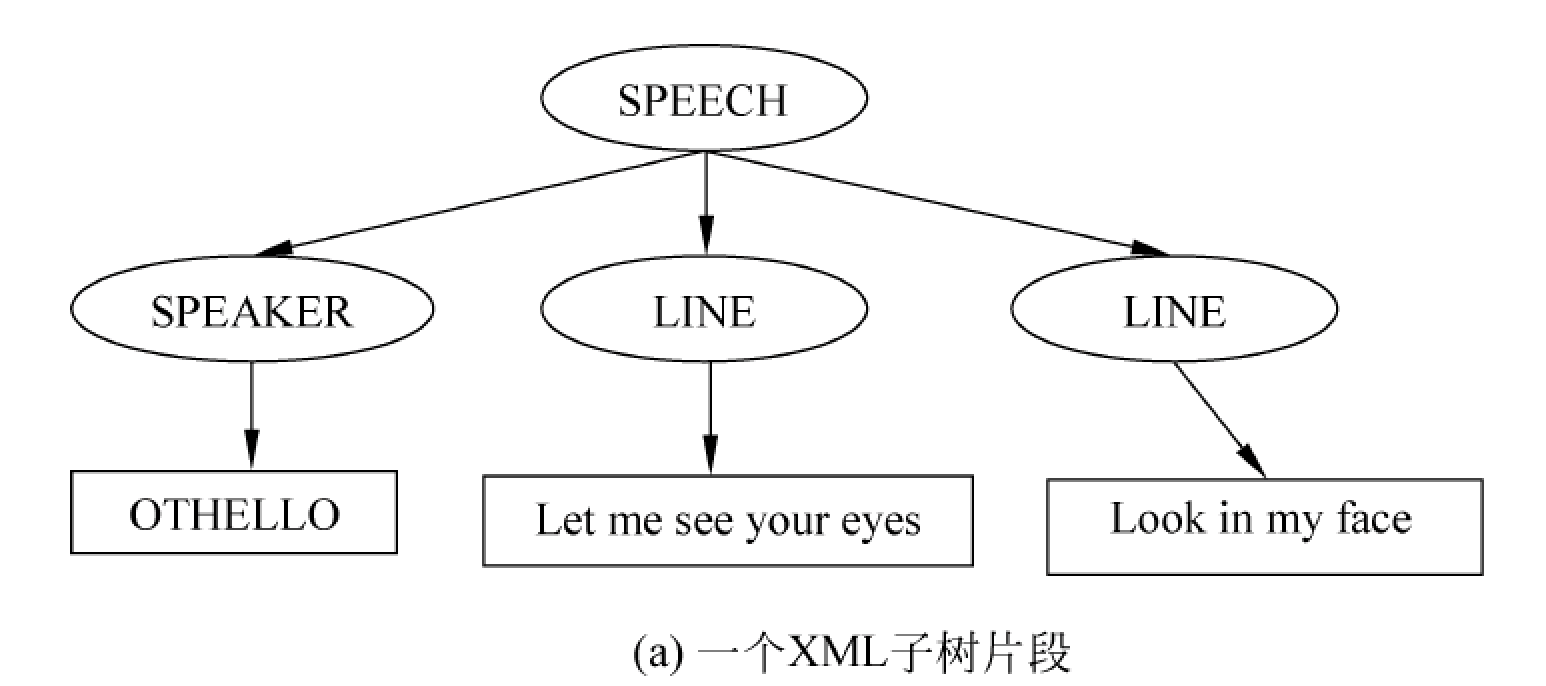
录聚集存储，即将类型相同的记录尽量存储在相邻的位置上。

目前，大部分纯 XML 数据库都采用深度优先的方法存储记录，因为深度优先实现起来比较简单，同时存储的顺序与 XML 文档的原始顺序是一致的。广度优先的方式还没有被采用，聚集存储的方法在 OrientX 中实现过，称为按逻辑意义的存储。所谓的逻辑意义，指语义，即语义相近的记录尽量聚集存储。

将记录粒度和记录存储顺序综合考虑，可以得到五种存储方法：结点级且深度优先存储(DEB)、结点级且同类聚集存储(CEB)、子树级且深度优先存储(DSB)、子树级且同类聚集存储(CSB)以及文档级存储(DB)。

文档级的存储方法实际上用得很少。文献[16]对其他四种存储方法进行了试验比较，结果如下：存储性能(主要涉及存储空间和物理页利用率)较好的是 CSB，查询性能(主要涉及 I/O 数量和 I/O 增长率)较好的是 DSB。

图 2-1 是 Natix 中记录的格式，其中(a)是一个 XML 子树片段，图(b)是这个子树作为一个记录存储时的内部表示。



(a) 一个XML子树片段

<table>
<tr><td>内容</td><td>10B</td><td>6B</td><td>6B</td><td>OTHELLO</td><td>6B</td><td>6B</td><td>Let…</td><td>6B</td><td>6B</td><td>Look…</td></tr>
<tr><td rowspan="3">结构</td><td rowspan="2">Header</td><td>Header</td><td>Header</td><td>Contents</td><td>Header</td><td>Header</td><td>Contents</td><td>Header</td><td>Header</td><td>Contents</td></tr>
<tr><td colspan="3">SPEAKER node</td><td colspan="3">LINE node</td><td colspan="3">LINE node</td></tr>
<tr><td colspan="10">SPEECH node</td></tr>
</table>

(b) 一个记录的内部表示

图 2-1　Natix 中记录的格式

从图中可以看出，结点是嵌套存储的，每个结点由两部分组成，一个是首部(header)，一个是正文(contents)。记录内的非根结点的首部长为 6B，

首部包含了 2B 的指向父结点的指针(有了这个指针后,结点其实可以在记录内移动;一个记录最大就是一个页,而一个页是小于 64KB 的,因而用 2B 的指针是足够的),还有 2B 表示对象的大小,另外的 2B 用来表示结点的类型(详见文献[18])。除了首部外,就是每个结点的正文信息。

2.2.2　纯 XML 数据库的索引技术

除了存储方案之外,索引技术也是决定一个数据库系统最重要的因素之一。讨论索引技术时,主要考虑两个问题:一是索引的对象,即在什么数据上建立索引;二是索引的组织结构。一般的纯 XML 数据库都对多个对象建索引,主要有值索引,即在属性值或结点内容上建索引;结点名索引,即在结点标记上面建立索引;边或路径索引,即在 XML 文档树的边上面建立索引,如路径字典(Path Dictionary)。

在 XML 数据库中,B^+ 树和哈希表仍然是最常用的索引结构。除此之外,也使用了一些新的索引结构,如 Trie、Patrica 等。这些索引结构一般都是针对 XML 数据的某些特征进行了优化,因而有可能发挥较好的性能。下面介绍几种代表性索引技术。

1. Fabric 索引

Fabric 索引[19,20]是一种全新的索引结构。基本思想是将半结构化数据之间的关系表示成路径,将路径编码成字符串,然后在这些字符串上建立一种索引结构。这种索引结构很适合复杂字符串的快速搜索,且代价比较低,已经引起了一些学者的兴趣。其特点是:

- 能够管理大量长而复杂的字符串;
- 其结构是平衡的,因而不同的访问所需的代价几乎是一样的;
- 本身的结构是高效的。虽然管理大量长字符串,但是由于采用了巧妙的算法组织这些字符串,因而数据的压缩率很高;
- 即可以作路径索引,也可以作值索引。

Fabric 索引是从 Trie 发展而来的。Trie 是字符串的一种编码方式。它

是一种多路树，用从根到叶的每条边表示一个字符。图 2-2(a)给出了一个 Trie 的例子。

Patricia Trie(简称 PT 树)是一种 Trie 的变体，Trie 中仅有一个孩子结点的内结点在 PT 树中没有表示出来。换个角度看，PT 树不是像 Trie 那样将每个字符都在边上表示出来，而是仅将字符串之间的差异表示出来。图 2-2(b)是一个 PT 树的例子，表示的串与图 2-2(a)完全相同。正是因为 PT 树只描述串之间的区别，因此它增长得很慢，同时索引项的长度对它的大小没有什么影响。

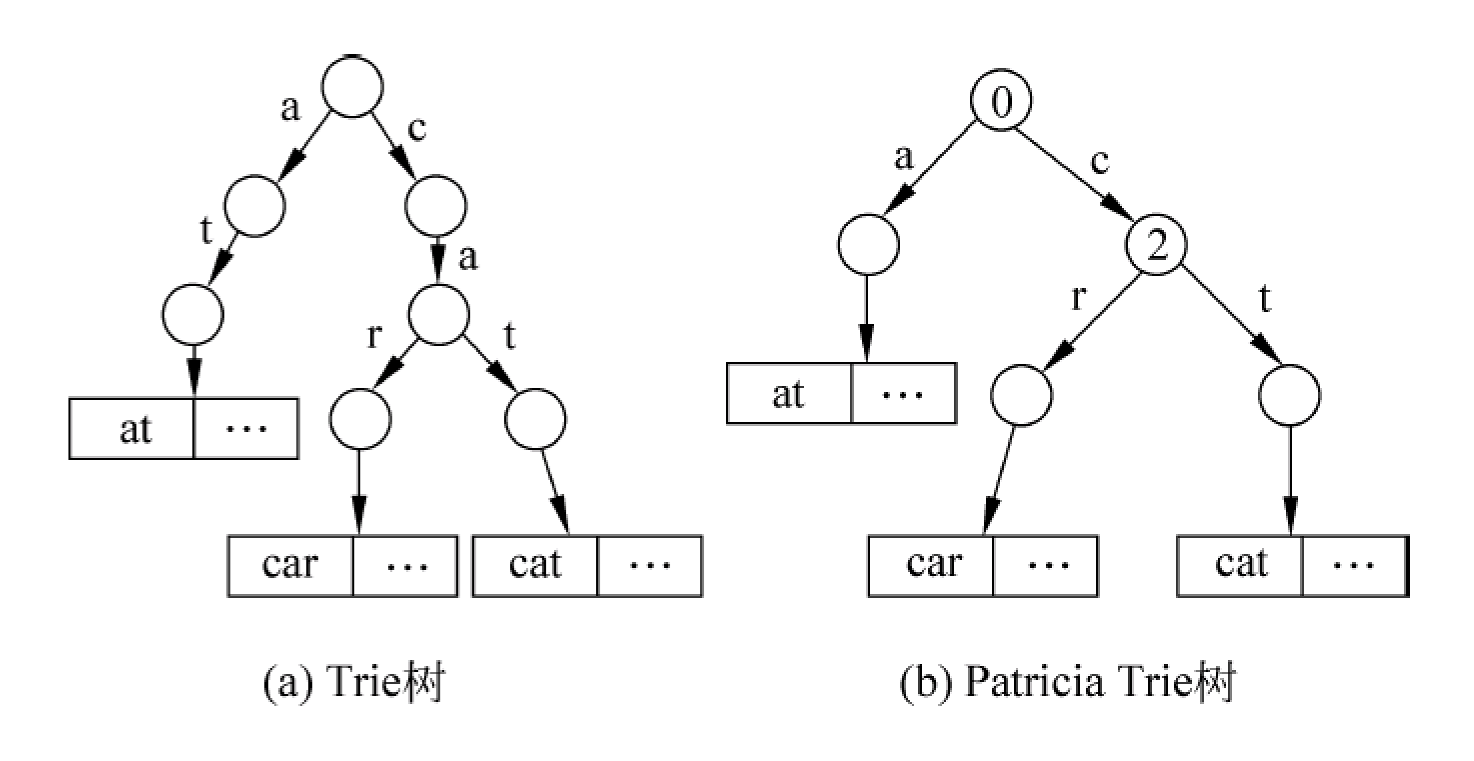


图 2-2　Trie 树与 Patricia Trie 树

PT 树本身是不平衡的，从图 2-2(b)中可以看出。Fabric 索引对 PT 树进行了平衡化处理。这种平衡化处理不是从 PT 树本身的纵向结构处理，而是通过附加一个水平的层次实现的。同时为了便于在计算机中存储，Fabric 索引把 PT 树分裂成若干个子树。每个子树存放在一个物理单位(如块 block)中，块与块之间仍然是相互关联的。普通的 Trie(以及 PT 树)在搜索字符串时都是从树的根结点出发自上而下遍历，直至到达叶结点。可以想象，如果树的层次太多，这种访问很耗时，并且在搜索不同的串时遍历的深度差别很大。在 PT 树中，解决这个问题的方法是构造一个高层(最底是 0 层，然后是 1 层……)，在第 i+1 层里包含了第 i 层的子树的指针，访问字符串时，就可以直接从高层出发，通过层与层之间的横向指针来直接访问相应的块，必要时再辅以纵向指针访问，从而大大节省访问的深度。这种分裂并平衡化处理后的 PT 树如图 2-3 所示。

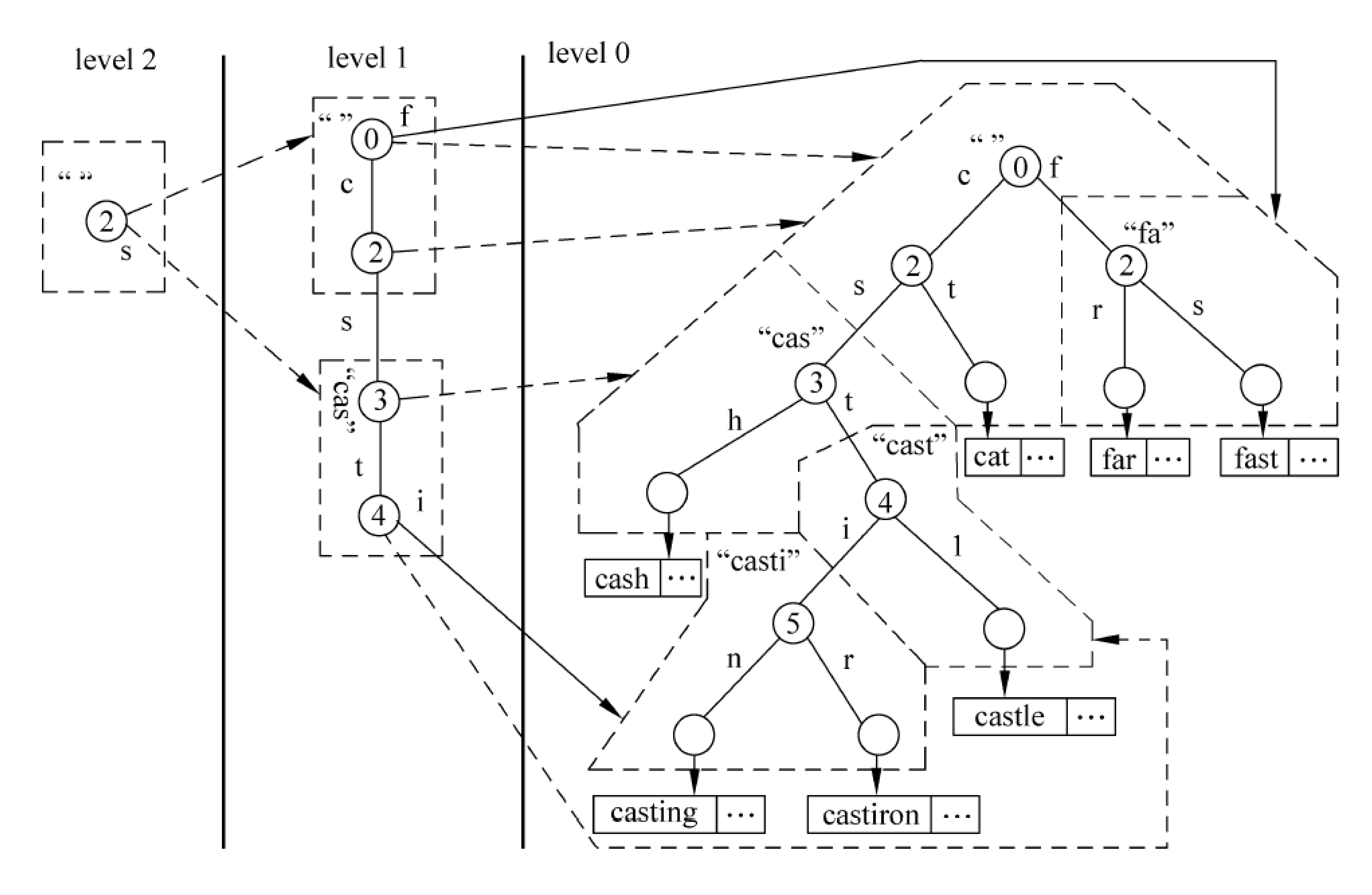


图 2-3　分裂并平衡化处理后的 PT 树

平衡化处理后，好像在水平方向上也构造了一个 PT 树，并且这个 PT 树的深度很小。现在对所有字符串的搜索都是从最高层（图 2-3 中的第 2 层）开始逐层深入，一直到最低层（图 2-3 中的第 0 层），因此这个过程总是平衡的，搜索不同的串所需的磁盘 I/O 总是一定的。

为了利用 Fabric 索引来管理 XML 的路径信息，文献[21]定义了两种编码路径的方式。一种是比较简单的，称为简单路径（raw path）；另一种包含了比较精确的信息，称为精致路径（refined path）。简单路径将从根到叶结点的路径按顺序编成字符串，如 XML 片段：

<A>alpha<B>beta<C>gamma</C></B></A>

有三条根部出发的路径<A>alpha、<A> <B>beta、<A><B><C>gamma，如果分别用 A、B、C 来代表<A>、<B>、<C>，那么这三条路径可以用字符串表示为 A alpha、A B beta、A B C gamma，这些串可以用 Fabric 索引管理以支持查询。精致路径则不仅对路径进行编码，也对其他信息进行编码，从而可以支持复杂查询，如含通配符的查询等。考虑图 2-4 所

示的 XML 文档。

```
<invoice>
    <buyer>
        <name> ABC Corp. </name>
    </buyer>
    <seller>
        <name> Acme Inc. </name>
    </seller>
    <item count=3> drill </item>
</invoice>
```

图 2-4 一个 XML 文档实例

如果查询“查询 X 公司卖东西给 Y 公司的发票”是一个频率很高的查询操作，那么可以构造一个精致路径：引入一个标志符 Z，然后将文档中对应的信息编码为“Z ABC Corp. Acme Inc.”，然后将这些串存储在 Fabric 索引中，那么这个串就对应图 2-4 中的文档，执行这个查询只需搜索这个索引结构，而不需操作原文档。

2. 相对区间坐标(relative region coordinate，RRC[22])索引

这种技术其实不仅是索引技术，更是一种存储技术。相对区间坐标是从绝对区间坐标(absolute region coordinate，ARC)演化而来的。所谓绝对区间坐标，实际上就是一种区间编码方案，即每个结点都联系一个二元组<start，end>，分别表示结点的开始和结束位置，并且都是在文档中的绝对位置。ARC 可以很好地支持查询操作，尤其是包含(containment)操作。对于更新操作，它的效率不是很好，原因就在于插入或更新很多结点的时候，可能会导致大量结点的区间坐标改变。RRC 就是针对更新操作在 ARC 基础上演变而来。一个结点的 RRC 坐标同样是一个二元组<start，end>，但是这两个坐标都是该结点在父结点中的相对位置，而不是绝对位置。在更新的时候，结点坐标的改变就可以局限在其父结点之内，从而提高更新效率。

这种方法的主要思路：一个结点更新(插入或删除)的时候，只会使有限

个结点受到影响。如果能把那些受到影响的结点聚集存储，比如，将这样的结点存储在一个块中，在更新的时候就只需读入这一个块，从而可避免过多的磁盘 I/O。另外，这些结点仍然以树的形式组织，并且保持它们的相对位置，子树之间也尽量保持原来的相对位置，从而可避免查询过于复杂。基于以上两个思想，可以把一个 XML 文档树划分为若干子树，即让结点聚集存储，每个子树的大小接近块大小，然后把这些子树块串起来，形成一棵树。在这种结构上进行查询和更新操作可以使二者的效率都可以接受。

很明显，基于这种结构进行查询时，每个结点的绝对位置都不能直接得到，而要经过多级运算，这样势必会降低查询效率。换句话说，这种方法只不过是更新效率和查询效率的一种折中。另外，这种方法的关键同样在于子树的划分是否合理，如果有大量插入操作，势必将引起子树的分裂，这种分裂的效率如何还需实验分析得知。

3. 实例分析

(1) Lore[23] 中的索引技术

在 Lore 中有四种索引。

- 链接索引 Lindex(link index)：Lore 基于一种称之为对象交换模型 OEM(object exchange model)的图状模型。Lore 本身没有提供访问双亲结点的机制，这个机制是用链接索引来实现的。链接索引以一个 oid(对象的 id)和一个标记 label 为输入，返回该结点的双亲结点，且该结点与双亲结点间的边标记为 label，如果 label 为空，则返回该结点的所有双亲结点。链接索引是用可扩展的哈希表实现的，因为链接索引永远只作等值检索。链接索引是系统自动根据整个数据库图产生的。
- 值索引 Vindex(Value Index)：值索引的对象是原子类型的对象。目前在 Lore 中主要是整型、实型和字符串型的对象。与一般索引项不同的是，值索引的索引项不只是这些对象的值，还包括一条入边的标记，因为有些标记名是经常进行查询的，如查询“book. price >

50”,在标记(price)和值(50)上建立索引,比单独在值上建索引要快。而一般的查询总是跟一定的标记联系起来的,当然并不是所有的值索引都包含标记,这是可选的。值索引在 Lore 中是用 B^+ 树索引实现的,所支持的数据类型只有三种,即整型、实型和字符串型。

- 文本索引 Tindex(Text Index):值索引可以快速支持一些简单查询,如等值查询和不等值查询等,但是当很多查询涉及到复杂的文本查询时,如关键字检索,值索引就力不从心了。为了支持更复杂的文本查询,Lore 中使用了文本索引。文本索引能够找到包含特定词的对象以及该词在对象值(文本)中的位置。与值索引类似,也可以在索引项中加入边标记。与传统的文本检索系统使用的索引结构相同,Lore 使用倒排列表来实现文本索引。
- 路径索引 Pindex(Path Index):Lore 的查询语言 Lorel 中经常用到路径表达式。如 DB. Movie. Title=‘Star Wars’,如果不用索引,这种路径查询就只能通过自上而下的遍历来实现,显然速度是很慢的。如果建立了路径索引,这种查询会快得多。路径索引返回通过路径能到达的所有对象。路径索引是和 Lore 的 DataGuide 配合使用的。

(2) Timer 中的索引技术

Timer 中的索引技术主要是从面向对象数据库中引进的。Timer 中的索引主要有两种:值索引和结点索引。值索引建立在属性值和元素结点内容上,当结点内容是打断文本时,将会建立基于词(term-based)的倒排索引(inverted index)。结点索引建立在元素结点标记名(tag name)上。这样,给定一个名称,就能够返回所有标记名为该名称的元素。Timer 中的所有索引都是建立在 Shore 的支持之上,都是用 B 树索引结构。

在关系数据库中,索引结构返回的是记录的 id(Rid),记录的 id 实际上指出了记录的物理存储地址。在 XML 数据库中,由于记录与结点不是对应的,因此不能直接返回记录的 id,而应返回结点 id。在 Timer 中,每个结点用唯一的三元组＜start,end,level＞来标识。start 和 end 分别是开始进入和离开结点的位置,并且总是满足 end 不小于 start。该结点的所有孩子和

后裔结点都在 start 和 end 之间，level 表示结点在文档中的层次。这种结构能够较好的支持双亲/孩子和祖先/后裔包含关系的判断，并且能较好的支持插入更新。

(3) Natix 中的索引技术

Natix 中同样使用了两种类型的索引，一种是单纯的文本上的索引，另一种是包含了某些结构信息的索引。

Natix 的文本索引使用的是一种很常用的索引结构—倒排文件(inverted file)。一般的倒排列表中只存储了文档的标识和文档内的偏移量，在 Natix 中，倒排列表中除了这些信息外还存储了一些额外的内容，这些内容包含了更多的上下文信息，同时尽量保证索引的性能，这样在查询时可以更精确，并且效率会更高。

除了文本索引外，Natix 中还引入了一种索引结构 XASR(Extended Access Support Relation)[24]。这种索引结构能够保持结点的双亲/孩子和祖先/后裔关系，在查询时特别有用。索引结构 XASP 是通过为每个结点都维持两个标记来实现的，这两个标记是 d_{min} 和 d_{max}，分别指深度优先遍历时第一次进入该结点(及其子树)和最后离开该结点时的值，实际上与 Timer 中的 start 和 end 有异曲同工之妙。在 XASR 表中，为每个结点都存储了一个表，表中的内容有结点 d_{min} 和 d_{max}、结点名、文档 id 以及父结点的 d_{min} 值。

XASR 是和文本索引配合使用的，XASR 能够快速判断结点间的包含关系，文本索引能够检索出特定词的位置。

2.3　使能 XML 数据库的存储结构和检索技术

使能 XML 数据库的特点是在原有的数据库系统上扩充对 XML 数据的处理功能，使之能适应 XML 数据存储和查询的需要。众所周知，关系数据库是使用最为广泛的数据库，为了充分发挥商用关系数据库系统数据处理能力强、查询性能好的优势，许多研究者正在从事基于关系数据库系统处理 XML 数据以及以 XML 文档发布关系数据的研究工作。

2.3.1 基于关系的 XML 数据存储

将 XML 文档映射为关系模式进行存储，有两大类映射方法：模型映射(model mapping)和结构映射(structure mapping)。对于模型映射，需要将 XML 文档模型(即文档树结构)映射为关系模式，关系模式表示 XML 文档模型的构造，对于所有 XML 文档都有固定的关系模式，因此，它与 XML 模式(或 DTD)无关；对于结构映射，需要将 XML 模式(或 DTD)映射为关系模式，关系模式用来表示目标 XML 文档的逻辑结构(即 XML 模式或 DTD)，它与 XML 模式(或 DTD)相关。

具体来说，利用关系数据库系统存储和查询 XML 数据有如下方法和策略[25-26]：

(1) 将一个 XML 文档看成是一个有序有向边标记图，称为 XML 图，设计一个(或若干个)关系存储 XML 图的边信息和结点值，该策略是属于基于边的模型映射方法，称为边模型映射方法。

(2) 设计若干个关系来存储 XML 文档树的结点信息、结点值和结构信息(通过区间编码来译码结构信息，或直接存储双亲/孩子结点或祖先/后裔结点对)，该策略是属于结点的模型映射方法，称为结点模型映射方法。

(3) 从 XML 文档的 DTD 或 Schema 推断 XML 元素应该怎样映射到关系表，该策略属于结构映射方法。

(4) 要求用户或系统管理者设计用于存储 XML 数据的关系表结构，对于关系表中的数据，可以直接以 XML 文档的方式进行发布[27]，也可以由用户或系统管理者使用 XML 查询语言或中间件提供的语言来定义该关系系统所对应的 XML 视图[28-31]。这样，其他应用就可以利用 XML 查询语言在虚的 XML 视图上构造一个查询，抽取 XML 视图中的数据片断并对抽取的部分进行物化，实现将关系数据转换为 XML 文档。该策略属于 XML-Enabled 数据库的方法。

1. 边模型映射方法

一个 XML 文档用一个有序有向边标记图(称为 XML 图)来表示[6]。在

这种图中，每一个 XML 元素用一个结点表示，结点被标上 XML 对象的 oid；元素与子元素(或属性)之间的关系用图中的边来表示，并在边上标上子元素(或属性名)；为了表示 XML 元素中各子元素的顺序，可以对图中从某结点引出的所有边进行排序；XML 文档中的值作为图中叶结点(即属性或最底层子元素结点)表示。图 2-6 是一个 XML 文档(图 2-5)的 XML 图，其中，标有数字的圆圈为非叶结点，数字为结点 oid；标有以“v”开头数字的圆圈为叶结点(即值结点)，圈中编号为结点 oid(称为 vid)；每条边的名称后面的数字为边的序号；4 号结点的 editorID 属性为 IDREF 类型，它直接指向 5 号结点。

```
<? Xml version="1.0"?>
<pub>
    <library>Beijing Library</ library >
    <book year="2000">
        <title>Database System Concepts</ title >
        <price>26.50</ price >
        <author id="101">
            <name>Kaily Jone</ name >
            <email>kjone@research.bell-labs.com</ email >
        </ author >
    </book>
    <book year="2001">
        <title>Introduction to XML</ title >
        <price>18080</ price >
        <author id="103">
            <name>Kaily Jone</ name >
        </ author >
    </book>
    <article editorID="105">
        <title>A Query Language for XML</ title >
        <author id="104">
            <name>Kaily Jone</ name >
        </ author >
    </ article >
    <editor id="105">
        <name>A. Deutsch</name>
    </editor>
</pub>
```

图 2-5　一个 XML 文档实例

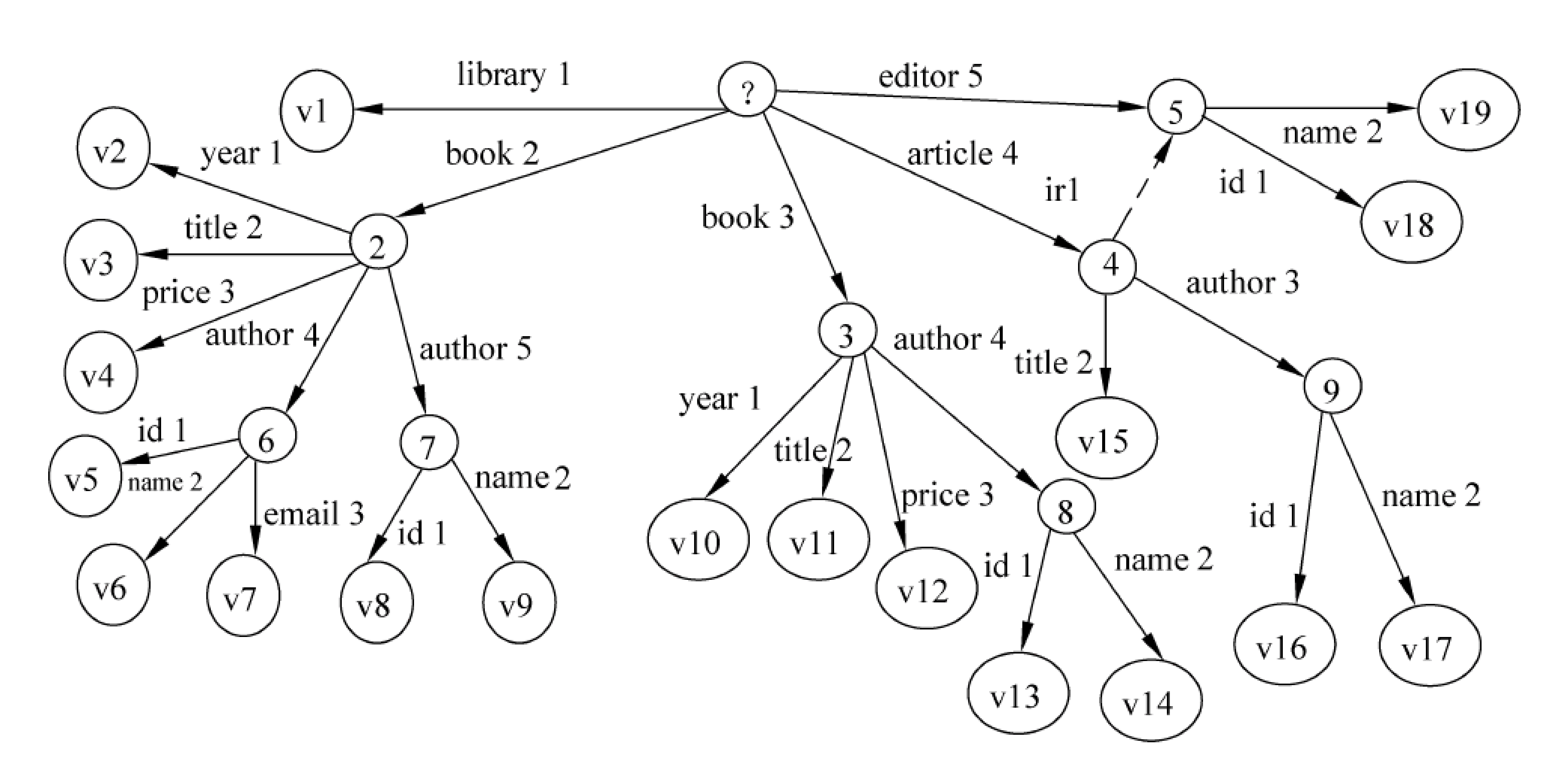


图 2-6 XML 图实例

有了 XML 图之后,就可以分别设计关系表存储 XML 文档的边信息和值。对于用来存储边信息的边表有三种设计方案:第一种是用一个表来存储图的所有边信息,这种方法称为 Edge 方法;第二种是所有具有相同名称的边放在一个边表中,这种方法称为 Binary 方法;第三种是采用一个边来存储图中所有路径的边信息,该方法称为 Universal 方法。

Edge 边表的关系模式为:Edge(source,ordinal,label,flag,target)。

其中,source 域和 target 域分别用来存储边的引出结点和引入结点对象的 oid,ordinal 域用来反映该边在兄弟边中的位置序号,label 域用来存储边标记(edge-label,即该边所指向结点的标记名),flag 属性用来反映该边所指向结点的类型,省略文档标识 docID。

结点的类型分为两类:叶结点和非叶结点。叶结点的类型分别为 integer、string 等,分别表示叶结点的值为整型、字符串型等;非叶结点的类型均为 ref。表 2-1 表示的是图 2-6 的 XML 图所对应的 Edge 方法的边表。

表 2-1 Edge 方法的边表

source	ordinal	label	flag	target
…	…	…	…	…
2	1	year	integer	v2
2	2	title	string	v3

续表

source	ordinal	label	flag	target
2	3	price	decimal	v4
2	4	author	ref	6
2	5	author	ref	7
…	…	…	…	…
4	1	editorID	ref	5
4	2	title	string	v15

Binary 边表与 Edge 边表的原理相同，只是将所有具有相同边标记的边存放在一个单独的边表中，Binary 边的关系模式为 Binary_{label}（source，ordinal，flag，target）。从概念上说，Binary 方法的边表是 Edge 边表的水平分割。

对于用来存储 XML 文档值的值表有两种设计方案：第一种是不单独设计值表，将值和边存储在同一个表中，在边表中直接增加一个属性 value，用于存储叶结点的值，这种方法称为内联方法；第二种是为每一种可能的取值类型设计一个值表，该方法称为分离值表。分离值表的关系模式为 Value_{type}（vid，value）。其中，vid 属性用来存储叶结点的 oid 值，value 属性用来存储叶结点的值。

2. 结点模型映射方法

Q. Li 和 B. Moon 首先对利用区间编码方案有效地处理 XML 的 RPE（Regular Path Expression）查询进行了研究[32]，以实现快速地确定在 XML 数据层次结构中任意结点对之间的包含关系。

1）XRel 模式

M. Yoshikawa 和 T. Amagara 等基于结点模型映射方法提出了一个 XML 数据的关系存储模型，称为 XRel[33]。XRel 是通过区间编码[start，end]来反映（译码）XML 文档的模型结构，并根据内容来划分边、属性边和文本边，同时将所有路径进行存储，因此 XRel 模式由以下 4 个关系表组成：

Element（pathID，docID，start，end，ordinal）

Attribute (pathID,docID,start,end,value)

Text (pathID,docID,start,end,value)

Path (pathID,pathexp)

其中,在 Path 表中,pathID 为标记路径(label-path)的标识,pathexp 域存储标记路径,为了实现路径表达式的字符串匹配操作,将标记路径中的“/”替换为“#/”进行存储。对于 Element、Attribute 和 Text 表,主键是(docID,start),pathID 是外键。

每一个不同的标记路径作为 Path 表的一个元组,因此它能够有效处理带“*”操作的正则路径表达式查询。第一步,利用字符串的匹配操作,能够快速地查找出与给定正则路径表达式相匹配的所有标记路径的标识;第二步,利用这些路径标识,能够快速地查找出隶属于这些路径终端的值(元素结点、文本结点或属性结点)。

基于 XRel 模式的查询实例如下:

对于 XPath 查询//book[price>=30.00]//name,对应的 SQL 查询为:

```
Select V2.value From Element E1, Path P1, Path P2, Path P3, Text V1, Text V2
Where P1.pathexp like '#%/book' and P2.pathexp like '#%/book#/price'
  And P3.pathexp like '#%/book/#%/name'
  And E1.pathID=P1.pathID        //找到所有满足"//book"的 book 元素
  And V1.pathID=P2.pathID        //找到所有满足"//book/price"的 price 值
  And V2.pathID=P3.pathID        //找到所有满足"//book/name"的 name 值
        /*包含关系检测条件*/
  And E1.start <V1.start and E1.end > V1.start
                                 //book 与 price 之间满足包含关系
  And E1.start <V2.start and E1.end > V2.start
                                 //book 与 name 之间满足包含关系
  And cast(V1.value as float)>=30.00
```

2) XParent 模式

香港科技大学 Jiang Haifeng、Lu Hongjun 和 Wang Wei 等基于结点模型映射方法提出了另一个 XML 数据的关系存储模式,称为 Xparent[34]。Xparent 通过一个单独的 Parent(parent-ID,child-ID)表来反映 XML 文档的模型结构,并根据内容和“结构与非结构”来划分边,同时将所有路径进行存储,因此,XParent 模式也由 4 个关系表组成:

LabelPah (pathID,length,pathexp)

Parent (pid,cid)

Element (pathID,did,ordinal)

Data (pathID,did,ordinal,value)

其中,length 为标记路径(label-path)的长度,即标记路径中边标记的个数;pathexp 域存储标记路径,与 XRel 类似,这里将标记路径中的“/”替换为“./”进行存储;did 为 XML 文档中元素结点的标识,它也可以作为以该结点为终端的数据路径(data-path)的标识;pid、cid 分别为 XML 文档的数据路径中的双亲结点、孩子结点的标识。

Parent 表存储的是双亲/孩子关系,因此,为了检查数据路径需要进行连接操作。为了加速这种处理,可以不用 Parent 表,而改用 Ancestor 表来存储祖先/后裔关系:

Ancestor (did, ancestor, level)

例如,元素(&9,&1,3)表示 &9 结点的曾祖父是 &1 结点。

利用 Ancestor 表能够快速地检测结点之间的祖先/后裔关系,但是它比 Parent 表需要更多的空间,而且由于存在冗余信息,修改起来代价也更高。

XParent 模式分别通过 LabelPath 表和 Parent 表来支持标记路径和数据路径,因此,XParent 模式既具有基于结点的模型映射的特点,又具有基于边的模型映射的特点。

Parent 表基于双亲/孩子关系来反映 XML 文档的核心结构,它也能够被进一步物化为 Ancestor 表来支持祖先/后裔关系。由于 XML 文档中的结点标识也可以用来标识以该结点为终端点的数据路径,因此,元素和数据(文本或属性)隶属于数据路径。

基于 XParent 模式的查询实例如下:

对于 XPath 查询//book[price>=30.00]//name,对应的 SQL 查询为:

```
Select D2.value
From LabelPath LP1, LabelPath LP2, Data D1, Data D2, Ancestor A1, Ancestor A2
Where LP1.pathexp like '.%/book./price' and LP2.pathexp like '.%/book.%/name'
   And D1.pathID=LP1.pathID      //找到所有满足"//book/price"的 price 值
```

```
And D2.pathID=LP2.pathID     //找到所有满足"//book//name"的 name 值
    /*结构关系检测条件*/
And D1.did=A1.did
And A1.level=1               //限定为 price 的双亲层
And D2.did=A2.did
And A1.ancestor=A2.ancestor  //price 的双亲是 name 的祖先(没有限定层次)
And cast(D1.value as float)>=30.00
```

结点模型映射方法需要维护的是 XML 文档树的结点信息，而不是边信息。XRel 模式通过区间编码来译码 XML 文档的模型结构(即包含关系)，它的优点是通过 θ 连接的代价比等值连接要高得多，这是因为在关系数据库中并没有特殊的索引机制来支持它。

XParent 模式在结构上类似于 XRel 模式，只是用 did 替代了[start，end]。然而，这种变化使得 XParent 模式仅需要等值连接，而 XRel 模式却需要 θ 连接，因此，XParent 模式能够基于传统的索引机制(如 B 树索引)，得到有效的实现。

3. 结构映射方法

J. Shamugasundaram 等提出了根据 DTD 映射关系模式的存储策略[35]，首先需要对 DTD 进行适当的简化，然后产生 DTD 图及元素图，再根据 DTD 图或元素图生成关系模式，最后将符合该 DTD 的 XML 文档数据装入关系数据库中。

1）简化 DTD 并生成 DTD 图

XML DTD 的元素说明可以是任意复杂的，例如，如下的元素说明是正确的。

<!ELEMENT a((b | c | e)?, (e ? | (f?,(b,b) *)) *)>　(2-1)

因此，在将 DTD 映射为关系模式之前，必须对一些复杂的 DTD 进行必要的简化变换。DTD 的简化变换包括三种类型。

① Flattening 变换：变换每一个嵌套定义到平坦的表示，使二元操作“,”和“|”不出现在任何操作之内，例如：

(e1,e2) * →e1 * ,e2 *　(e1,e2) ? →e1?,e2?　(e1| e2) → e1?,e2?

② Simplification 变换：将连续的多个一元操作缩简为一个一元操作，例如：

e1 * * → e1 *　　e1 * ? → e1 *　　e1 ? * → e1 *

③ Grouping 变换：将多个具有相同名称的子元素进行聚合，形成一个子元素，例如：

…,a * ,…,a * ,…→a * ,…　　…,a * ,…,a?,…→ a * ,…

…,a?,…,a * ,…,→ a * ,…　　…,a?,…,a?,…→ a * ,…

…,a,…,a,…→ a * ,…

另外，所有的“＋”操作被转换成“ * ”操作。例如式(2-1)可以被简化变换为：

$$<!\ ELEMENT\ a(b*,c?,e*,f*)> \quad (2\text{-}2)$$

一个 DTD 图表示一个 DTD 的结构，图的结点是 DTD 中的元素、属性和正则路径运算符，每一个元素在图中出现且仅出现一次，属性和操作符在 DTD 图中出现的次数与它们在 DTD 中出现的次数相同；图的边则反映 DTD 中元素之间的嵌套关系；图中的环表示回路的出现。图 2-7 显示的是根据一个 DTD 所产生的 DTD 图，其中，斜体字表示的是根据属性生成的结点。

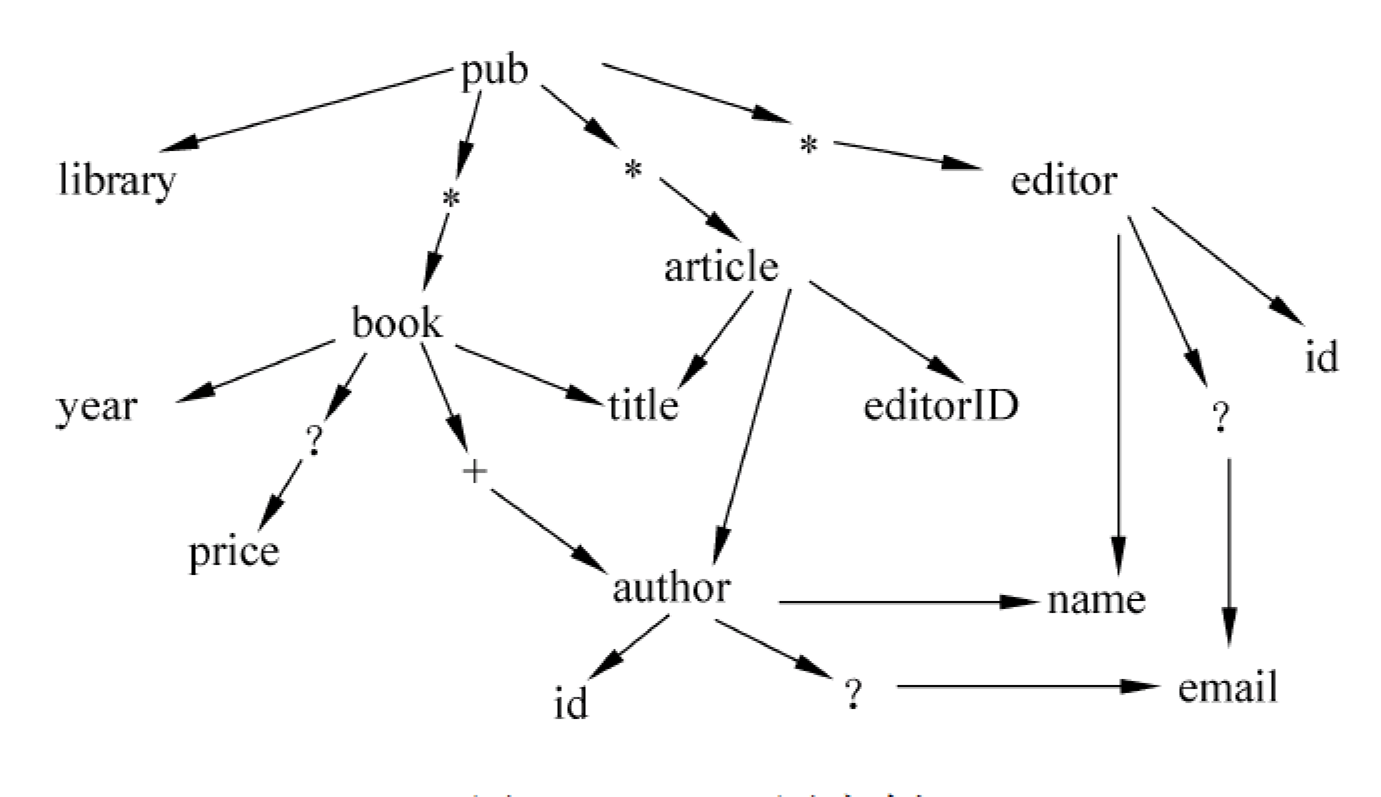


图 2-7　DTD 图实例

2）根据 DTD 图生成关系模式

根据一个 DTD 图生成的关系模式是该 DTD 图中每一个元素所生成的关系模式的并。为了说明如何生成一个元素的关系模式，需要引入元素图

的概念。简单地说，一个元素的元素图就是从该元素出发以深度优先遍历 DTD 图的过程中所生成的一棵树。如果遍历过程中到达了一个已经遍历过的结点，则表示出现了回路，回路可看成逆向边，以虚线边表示，且该条路线不必再去重复遍历。

根据 DTD 图生成关系模式的方法有：基本内联法、共享内联法和综合内联法。

(1) 基本内联法

基本内联法根据每一个名为 e 的元素的元素图生成一个同名关系 e，元素图中根结点 e 的所有后裔叶结点均内联到该关系中来作为一个属性。但下列两种情况除外：

① 一个 * 或＋结点的直接孩子结点不包括在该关系中，即对于集合值孩子将另外生成一个新的关系；

② 产生逆向边的结点不包括在该关系中，即另外生成一个新的关系来处理回路。

在生成的关系模式中，关系的属性是以从结点 e 到内联结点的路径来命名的；每一个关系需要有一个 ID 域来作为该关系的键；对于 * 或＋结点的直接孩子结点以及带回路的结点，它们所生成的关系还需要有一个 parentID 域来作为该关系的外键。基本内联法将产生大量的关系，并且会产生大量的数据冗余，因此该方法基本上不实用的。

(2) 共享内联法

共享内联法使每一个元素结点出现且只出现在一个关系中。它按如下原则来判断哪些元素可以生成独立的关系：

- DTD 图中入度大于 1 或等于 0 的元素结点生成独立的关系(模式)；
- DTD 图中结点 * 或＋的直接后继元素结点生成独立的关系(模式)；
- 互为递归的入度均为 1 的元素结点，其中之一生成独立的关系(模式)；
- 其余的结点均生成关系属性。

图 2-8 显示的是根据图 2-7 所示 DTD 图按共享内联法所产生的关系模

式。共享内联法与基本内联法相比减少了数据冗余,但是在查询时却增加了许多连接操作。

```
Pub(pubID:integer, pub.library.isroot:boolean, pub.library:string)
Book(bookID: integer, book.parentID: integer, book.parentCODE: integer,
book.year:string, book.price.isroot:boolean, book.price:string)
article (articleID: integer, article.parentID: integer, article.parentCODE:
integer, article.editorID:string)
editor (editorID: integer, editor.parentID: integer, editor.parentCODE:
integer, editor.id:string)
title(titleID:integer, title.parentID:integer, title.parentCODE;integer, title:
string)
author (authorID: integer, author.parentID: integer, author.parentCODE:
integer, author.id:string)
name(nameID:integer, name.parentID:integer, name.parentCODE:integer,
name:string)
email(email ID:integer, email.parentID:integer, email.parentCODE:integer,
email:string)
```

图 2-8　共享内联法产生的关系模式

将元素 X 内联到元素 Y 所产生的关系中,如果有一个 XML 文档是以 X 元素为根,则在查询时会产生问题。为了解决这种元素的查询,在关系中引入一个 X.isroot 域,如 book 关系中的 book.price.isroot 域。

在 XML 文档中,有些元素的双亲并不是唯一的。例如,author 元素的双亲可能是 book 元素,也可能是 article 元素。因此,在非根元素所产生的关系中还需要增加一个 parentCODE 域,以指示双亲的类型,以便查询时进行正确的连接。

(3) 综合内联法

在共享内联法的基础上,将所有入度大于 1 的元素结点也内联进入父结点所生成的关系中,但是带回路的结点以及结点 * 和 + 的直接后继结点除外。综合内联法的出发点是充分汲取基本内联法和共享内联法的优点,克服其缺点。图 2-9 显示的是根据图 2-7 所示 DTD 图按综合内联法所产生的关系模式。

```
Pub(pubID:integer, pub.library.isroot:boolean, pub.library:string)
Book(bookID: integer, book. parentID: integer, book. parentCODE: integer, book.year:string,
    book. price. isroot: boolean, book. price: string, book. title. isroot: Boolean, book.title:string)
article (articleID: integer, article. parentID: integer, article. parentCODE: integer, article.editorID:string,
    article. title: string, article. author. id: string, article. author. name: string, article.author.email:string)
editor (editorID: integer, editor. parentID: integer, editor. parentCODE: integer, editor.id:string,
    editor.name:string, editor.email:string)
author (authorID: integer, author. parentID: integer, author. parentCODE: integer, author.id:string,
    author.name.isroot:Boolean, author.name:string, author.email.isroot: Boolean, author.email:string)
```

图 2-9 综合内联法产生的关系模式

对于 XPath 查询/pub/article[title="A Query Language for XML"]/author/name,基于共享内联法所对应的 SQL 查询为:

```
select N.name
from pub P, article AR, title T, author AU, name N
where P.pubID=AR.[article.parentID]
  and AR.articleID=T.[title.parentID]
  and T.[title.parentCODE]=1
  and AR.articleID=AU.[author.parentID]
  and AU.[author.parentCODE]=1
  and AU.authored=N.[name.parentID]
  and N.[name.parentCODE]=0
  and T.title='A Query Language for XML'
```

其中,条件 T.[title.parentCODE]=1 隐含着仅查找 article 元素(非 book 元素)的 title 子元素;条件 AU.[author.parentCODE]=1 隐含着仅查找 article 元素的 author 子元素;条件 N.[name.parentCODE]=0 隐含着仅查找 author 元素(非 editor 元素)的 name 子元素。

(4) 共享内联法与综合内联法的比较

共享内联法与综合内联法各有优缺点,如上面的实例所示,综合内联法

可以减少 SQL 查询中连接操作的数量。但是,共享内联法通常需要的 SQL 子查询更少。例如,对于//title、//name、//aythor/name 之类的查询,综合内联法需要多个 SQL 子查询,而共享内联法仅需要一个 SQL 查询。

2.3.2 X-RESTORE 数据模型

一个 XML 文档可以被建模为一棵树,因此,一个 XML 文档的集合可以被建模为一个森林(forest)。一个根结点是文档结点的树指向一个 XML 文档,一个根结点是非文档结点的树指向一个 XML 文档片段(fragment)。因此,一个 XML 查询既可以在一个 XML 文档树或文档森林上进行,也可以在一个 XML 文档片段或森林片段上进行。

XML 文档中的字符数据有两种类型:文本结点和简单类型值(simple-typed values)。一个文本结点表示一个连续的字符信息项的字符串,并且不允许其他文本结点作为它的直接兄弟结点。简单类型值是一个 XML 模式的简单类型和该类型的值的总称。一个叶子元素结点的文本内容可以被说明为一个简单类型值,例如:一个整数 57、一个日期 10/01/02、一个单价的序列(21.5,23.6,28.3)。查询数据模型逻辑上同时支持文本结点和简单类型值,但它并没有指定两者都必须在物理上实现。一般情况下,物理上的实现可以仅选择文本结点,并按要求来构造简单类型值。X-RESTORE 就是按照这种方式实现的,即所有 XML 的叶子元素结点的内容或属性结点的值都作为字符串类型进行存储,需要时再将它们按要求转换为简单类型值,如整型、日期型或简单类型的序列等。

下面给出一个 XML 文档树模型的形式化定义。这里,忽略了命名空间结点、处理指令结点和注释结点。

定义 2-1　一个 XML 文档是一个有序树,记为 $T=\langle V,E,r,\text{label},\text{rank}\rangle$。其中,$V$ 是所有结点的集合,$E\subseteq V\times V$ 是边的集合,$r\in V$ 是文档根结点,并且

(1) $V=r\cup V_E\cup V_A\cup V_T$,其中 V_E、V_A 和 V_T 分别表示元素、属性和文本结点的集合。

（2）$E=E_r\cup E_E\cup E_A\cup E_T$，其中 $E_r\subseteq\{r\}\times V_E$ 是根边的集合，$E_E\subseteq V_E\times V_E$ 是元素边的集合，$E_A\subseteq V_E\times E_A$ 是属性边的集合，$E_T\subseteq V_E\times V_T$ 是文本边的集合。这里，根边是文档根结点 r 指向（根）元素结点的边，元素边、属性边和文本边分别是从元素结点指向子元素结点的边、指向属性结点的边和指向文本结点的边。元素边与文本边又统称为孩子边。

（3）$\text{label}=\text{label}_\text{E}\cup\text{label}_\text{A}\cup\text{label}_\text{T}$。

① 函数 label_E：$V_E\rightarrow$<integer，string，string>。给每个元素结点赋予一个三元组，分别表示该元素结点的对象标识、元素标记名和结点类型。结点类型的值是 EE、ET、EM 和 EN，它们分别代表元素结点的内容是子元素、文本、子元素与文本的混合或空元素。

② 函数 label_A：$V_A\rightarrow$<integer，string，string，string >。给每个属性结点赋予一个四元祖，分别表示该属性结点的对象标识、属性名、属性值（字符串值）和属性值的类型。

③ 函数 label_T：$V_T\rightarrow$<integer，string，string >。给每个文本结点赋予一个三元组，分别表示该文本结点的对象标识、文本结点的内容（字符串值）和内容的类型。

（4）函数 rank：$V\rightarrow$integer。给每个结点赋予一个整数，表示该结点的文档序号。结点的文档序号在一个文档内唯一，它反映了在先序遍历文档树时该结点的出现次序。对于一个元素结点而言，这里规定它所拥有的属性结点的文档序号必须小于它的内容结点（子元素结点和文本结点）的文档序号。由于元素结点中所包含的各个属性结点之间不区分次序，因此，对于某元素结点所包含的各属性结点的文档序号的先后无关紧要。

例如，对于图 2-10 所示的 XML 文档片段，第 2 节第 1 条的定义数据模型如图 2-11 所示。假设对 XML 文档（树）进行先序遍历，每个结点得到一个先序遍历序号，可以将此先序遍历序号作为结点的对象标识。由于结点的先序遍历序号已经反映了结点之间的文档顺序，因此，结点的先序遍历序号同时也可作为结点的文档序号。

```
<book id=1658 year=2003>
<title>XML Databases</title>
<price currency="RMB" money=32.5/>
<abstract>Researches on XML database system primarily include <bold>
index structure</bold>
    of XML data, efficient implementation of <bold>structural joins </
bold>and optimization techniques of XML queries etc.
</abstract>
</book>
```

图 2-10　XML 文档片断

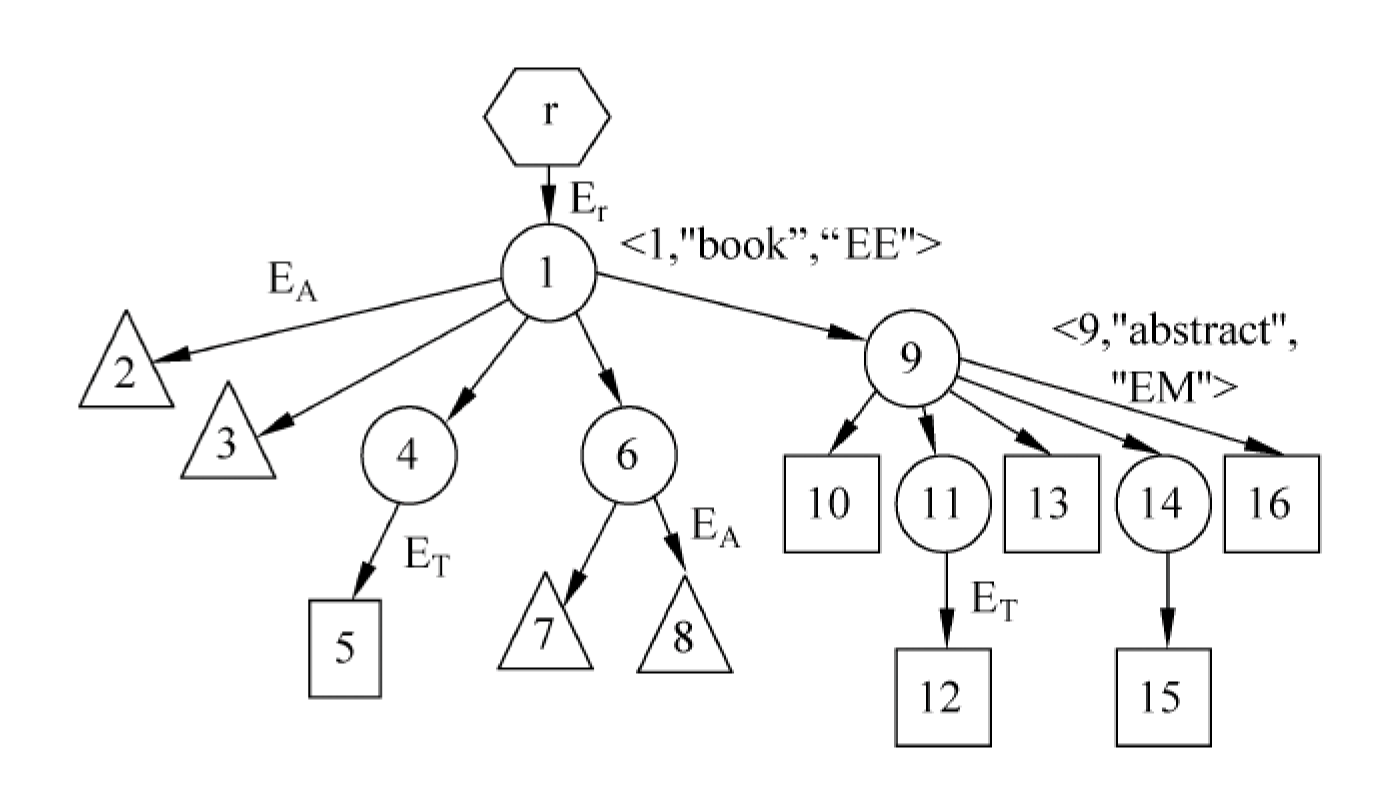


图 2-11　XML 文档片断的数据模型实例

这里，元素结点的集合 $V_E=\{1,4,6,9,11,14\}$，属性结点集合 $V_A=\{2,3,7,8\}$，文本结点的集合 $V_T=\{5,10,12,13,15,16\}$；根边的集合 $E_r=\{r\to 1\}$，元素边的集合 $E_E=\{1\to 4,1\to 6,1\to 9,9\to 11,9\to 14\}$，属性边的集合 $E_A=\{1\to 2,1\to 3,6\to 7,6\to 8\}$，文本边的集合 $E_T=\{4\to 5,9\to 10,9\to 13,9\to 16,11\to 12,14\to 15\}$；函数 label 的映射结果如表 2-2 所示。

2.3.3　一种基于关系的 XML 数据索引和查询

为了有效实现对文档树中任意两个结点对之间的祖先/后裔关系、双亲/孩子关系以及文档位置关系的检测，以加速路径表达式的计算，同时实现按关键字搜索 XML 文档。下面给出一种 XML 数据的索引结构，称为扩展先序列表(Extended Preorder List)。其基本思想是：对 XML 文档树中

表 2-2　函数 label 的映射结果

结点类型	结点编号	函数 label 的映射结果
元素结点	1	<1,"book","EE">
	4	<4,"title","ET">
	6	<6,"price","EN">
	9	<9,"abstract","EM">
	11	<11,"bold","ET">
	14	<14,"bold","ET">
属性结点	2	<2, id,"1658" , ID>
	3	<3, year,"2003", integer>
	7	<7, currency,"RMB", string>
	8	<8, money,"32.5", decimal>
文本结点	5	<5,"XML Databases", string>
	10	<10," Researches on XML database system primarily include", string>
	12	<12,"index structure", string>
	13	<13,"of XML data, efficient implementation of", string>
	15	<15,"structural joins", string>
	16	<16," and optimization techniques of XML queries etc.", string>

的所有对象进行先序遍历(即深度优先遍历),产生这些对象的扩展先序遍历序号,并将这些对象的扩展先序遍历序号按升序进行列表。XML 文档树中的对象包括元素结点、属性结点以及属性值和文本结点内容中的“关键字”。下面介绍具体的处理方法。

(1) 对于所有具有相同元素标记名 Tag 的元素结点建立一个元素索引表(即扩展先序列表)Elem_Tag,该索引表中的每一个记录存储该结点的一个八元组<docID, order, maxOrder, depth, pathID, parentOrder, parentMax, ssIndex>。其中,docID 是该结点所在文档的文档标识;order 是该结点的扩展先序遍历序号;maxOrder 是以该结点为根的子树中所包含的所有元素结点、属性结点以及文本结点和属性值中的关键字的最大 order,以反映结点之间的祖先/后裔关系;depth 是该结点在文档树中所处的层次,以反映祖先/后裔关系中的嵌套层数关系;pathID 是从根开始到该结点的路径表达

式的标识，以支持对简单绝对路径表达式按路径索引的方法（即路径法）进行计算；parentOrder 和 parentMax 分别是该结点的双亲元素结点的 order 和 maxOrder，parentOrder 用来有效地支持结点之间的双亲/孩子关系、左兄弟/右兄弟关系的计算，parentMax 用来加速结构连接的计算；ssIndex 是该结点在它的同名兄弟结点中所处的位置序号，以便有效地实现按位置查询同名兄弟的操作。

（2）对于所有具有相同属性名 Name 的属性结点建立一个属性索引表 Attr_Name，该索引表中的每一个记录存储该结点的一个七元组<docID, order, maxOrder, depth, pathID, parentOrder, parentMax>。

为了便于文档的修改，结点的 order 取值并不是连续的，要为它的可能的后裔对象的插入预留序号的空间，因此被称为“扩展”先序遍历序号。同时 MaxOrder 的取值也不要求是以该结点为根的子树中所包含的所有元素结点、属性结点以及文本结点和属性值中关键字的最大 order，它的取值可以是一个不小于该结点为根的子树中所包含的所有对象的最大 order 的任意一个整数；该结点之后的第一个结点的 order 值只要大于该结点的 MaxOrder 值即可。

（3）对于所有属性值和文本结点中出现的每一个关键字 Keyword 建立一个关键字索引表 Word_Keyword，该索引表中的每一个记录存储该关键字的一个三元组<docID, order, depth>。其中，docID 是该关键字所在文档的文档标识；order 是该关键字的扩展先序遍历序号；depth 是该关键字在文档树中所处的层数，以反映它是直接属于哪一个属性结点或双亲元素结点。

关键字索引表 Word_Keyword 的作用是有效地实现按关键字搜索。同样，为了提高处理性能，按相同的关键字 Keyword 分别组织一个关键字索引表。

（4）对于 XML 文档中所有从根开始到另一个元素或属性的路径建立一个路径索引表 Path，该索引表中的每一个记录存储该路径的一个二元组<pathID, pathExp>。其中 pathID 是该路径的标识；pathExp 是该路径的

路径表达式，这种路径表达式是形如/name1/name2/…/namek 或/name1/name2/…/@namek 的字符串。为了实现在 RDBMS 中对一个简单绝对路径表达式进行串匹配操作，它的存储形式用“#/”来代替“/”，并假设属性结点的值或文本结点的内容的路径表达式与对应的属性结点或双亲元素结点的路径表达式相同。路径索引的作用是加速长的简单绝对路径表达式的计算。

(5) 还要建立一个 Document 索引表，索引表的每一个记录与一个 XML 文档相对应，用来存储 docID（文档标识）、URL（文档所在位置的 URL）以及其他一些与该文档相关的信息；一个 Name 索引表，索引表的每一个记录与 XML 文档中的一个元素标记或属性名相对应，用来存储 nameID（名字标识）、name（标记名或属性名）以及其他一些与该名字相关的信息。

对于 XML 文档森林中的任意一个结点 n，region(n) 表示区间[order(n)，MaxOrder(n)]；任意一个词 w，region(w) 表示区间[order(w)，MaxOrder(w)]。

利用 XML 数据的索引结构，文档树中任意两个对象之间的祖先/后裔、双亲/孩子关系，任意两个元素结点之间的之前/之后关系、左兄弟/右兄弟关系，分别可以通过命题 2-1、命题 2-2、命题 2-3 和命题 2-4 直接进行判断。

命题 2-1 对于一个 XML 文档树中的任意两个对象 x 和 y，对象 y 是对象 x 的后裔，当且仅当 region(x) contains region(y)。这里 region(x) contains region(y) 意味着 docID(x)＝docID(y) and order(x)＜order(y) and order(y)＜＝maxOrder(x)。

命题 2-2 对于一个 XML 文档树中的任意两个对象 x 和 y，对象 y 是对象 x 的孩子，当且仅当 docID(x)＝docID(y) and order(x)＝parentorder(y)，或者 region(x) directly contains region(y)（即 region(x) contains region(y) and depth(x)＝depth(y)－1）。

命题 2-3 对于一个 XML 文档树中的任意两个元素结点 x 和 y，按文档顺序，结点 x 是位于结点 y 之前（或之后），当且仅当 docID(x)＝docID(y)

and maxOrder(x)＜order(y)(或者 order(x)＞ maxOrder(y))。

命题 2-4　对于一个 XML 文档树中的任意两个元素 x 和 y，按文档顺序，结点 x 是位于结点 y 之前(或之后)的兄弟结点，当且仅当 docID(x)＝docID(y) and maxOrder(x)＜order(y)(order(x)＞ maxOrder(y) and parentOrder(x)＝parentOrder(y))。

图 2-12 所示的是一个 XML 文档片断和它的 DTD 片断；表 2-3 反映的是该 XML 文档的扩展先序列表的编码结果，包括元素和属性结点的(order，maxOrder，depth)以及关键字的(order，depth)的编码结果。

```
<book editor= "Tom Smith ">
    <title>Database System Concepts</title>
    <author id= "102 ">
        <name>Kaily Jone</name>
    </author>
    <author id= "103 ">
        <name>Silen Smith Jone</name>
    </author>
</book>
```

(a) XML文档片段

```
<!ELEMENT book(title, pres  s?, author+)>
<!ATTLIST book editor CDATA IMPLIED>
<!ELEMENT author(name, contact?)>
<!ATTLIST author id #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT press (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT contact (#PCDATA)>
```

(b) DTD片段

图 2-12　XML 文档片断和它的 DTD 片断

表 2-3　XML 文档片断的扩展先序列表的编码

(a) 元素或属性结点的(order，maxOrder，depth)编码

元素或属性名	(order，maxOrder，depth)
book	(1,150,0)
editor	(2,6,1)
title	(7,27,1)
author	(39,66,1)
id	(40,40,2)
name	(41,45,2)
author	(67,94,1)
id	(68,68,2)
name	(69,73,2)

(b) 关键字的(order，depth)编码

关键字	(order，depth)
Tom	(3,2)
Smith	(4,2)
Database	(8,2)
System	(9,2)
Concepts	(10,2)
Kaily	(42,3)
Jone	(43,3)
Silen	(70,3)
Smith	(71,3)
jone	(72,3)

图 2-13 所示的是设计用来存储 XML 文档的关系模式，称为 X-RESTORE(XML'S Relational Storage and Retrieval)[36]。具体说明如下：

Document(docID, URL, …)
Name(nameID, name, …)
Path(pathID, pathExp, …)
Elem_tag (docID, order, maxOrder, depth, attrID, nodeType, pathID, parentOrder, parentMax, ssIndex)
Attr_Name(docID, order, maxOrder, depth, pathID, parentOrder, parentMax)
Word_keyword(docID, order, depth)
Value(docID, order, afterOrder, value, valueType, pathID)
Structure(docID, order, maxOrder, depth, nodeType, nameID, parentOrder, parentMax)

图 2-13　XML 文档的关系存储模式

(1) 对于 Elem_Tag 表，nodeType 列存储的是元素结点的类型，它的取值分别是 EE、ET、EM 或 EN，分别代表元素结点的内容是子元素、文本、子元素与文本的混合或空元素；attrID 列存储元素结点的 ID 类型的属性值，这样安排是为了有效地实现按 ID 属性来精确定位，如有效地实现解除引用(dereference)操作“＝＞”等。

(2) Value 表用来存储 XML 文档的内容，即所有文本结点的内容及属性结点的值。在 X-RESTORE 中，所有属性值和文本内容都是以字符串的形式进行存储的，但查询经常需要转换字符串到其他的带有更多语义信息的数据类型。因此，Value 表中的 valueType 列就是用来说明属性值或文本内容的原本数据类型的。Order 列存储包含该属性值或文本内容的属性结点或元素结点的 order。对于 EM 类型的元素结点而言，一个序号为 order 的元素结点可能包含多个文本内容串，但每一个文本内容必然被安排在本元素结点或它的某个子元素结点之后出现，因此，afterOrder 列就是存储这种用于标识文本内容串所在位置的元素结点序号。

(3) Structure 表用来存储所有元素结点和属性结点的扩展先序列表，即文档的总索引。其中，通过 nameID 列可以从 name 表中获得每一个元素结点的标记名和属性结点的属性名；nodeType 列存储的是结点类型，它除了上述已说明的元素结点类型的取值之外，再加上一个取值 A，表示结点的

类型是属性结点。建立该表的目的是为了有效地实现 XML 文档的结构查询以及查询结果的文档树片段的重构。例如，对于一个给定的元素，查找它的所有孩子、后裔或属性结点。

(4) 关系表中带下划线的属性为码属性。关系表都按码属性建立聚集索引(即主索引)，以加快查找的执行速度。另外，关系表 Elem_Tag、Attr_Name 和 Value 还要按 pathID 建立辅助索引；关系表 Elem_Tag 还要按(docID，attrID)建立辅助索引。

在对 XML 的简单绝对路径表达式进行查询处理时，首先将查询路径表达式中的“/”和“//”分别用“#/”和“#%/”进行替换；然后将替换后得到的路径表达式在 path 表的 pathExp 列上进行字符串匹配；最后将匹配得到的结果与路经表达式中最后一个元素或属性所对应的 Elem_Tag、Attr_Name 或 Value 等关系进行等值连接。这种表达式的查询计算方法为路径法。

2.4　纯 XML 数据库和使能 XML 数据库技术的比较

使能 XML 数据库主要是以关系数据库为主，因此我们首先对传统的关系数据库和 XML 数据库进行对比。关系数据库的逻辑结构是单一的关系表，而 XML 数据库的逻辑结构是树；关系数据库中涉及的主要操作是表的连接，而 XML 数据库中的主要操作是路径的匹配；关系数据库中的存储粒度就是元祖，而 XML 数据库中的存储粒度却是多种多样，常见的有结点级、子树级等；关系数据库中数据的更新一般只会影响被作用的记录，而 XML 数据库中数据的更新除了会影响当前记录外可能还会影响其他记录(比如父结点的结构改变，势必会改变子结点的结构)；关系数据库中对数据的更新就是修改其内容，在 XML 数据库中既有对结点内容的更新，也有对文档结构的更新；关系数据库中对数据的访问总是遵循表-元祖-属性的顺序，而 XML 数据库中对数据的访问一般是自上而下路径导航，但也有时是自下而上(考虑 XPath 的轴 ancestor 和 parent)，有时是平行的(如 XPath 的 following-sibling 轴等)，有时又是毫无特征的转跳(考虑 XML 文档的

IDREF 属性)等。总之关系数据库与 XML 数据库在结构和操作方面有很多不同,使得纯 XML 数据库和使能 XML 数据库在实现技术上存在差异,主要体现在以下几个方面:

1. 系统体系结构的不同

一个典型的纯 XML 数据库系统的体系结构如图 2-14 所示。

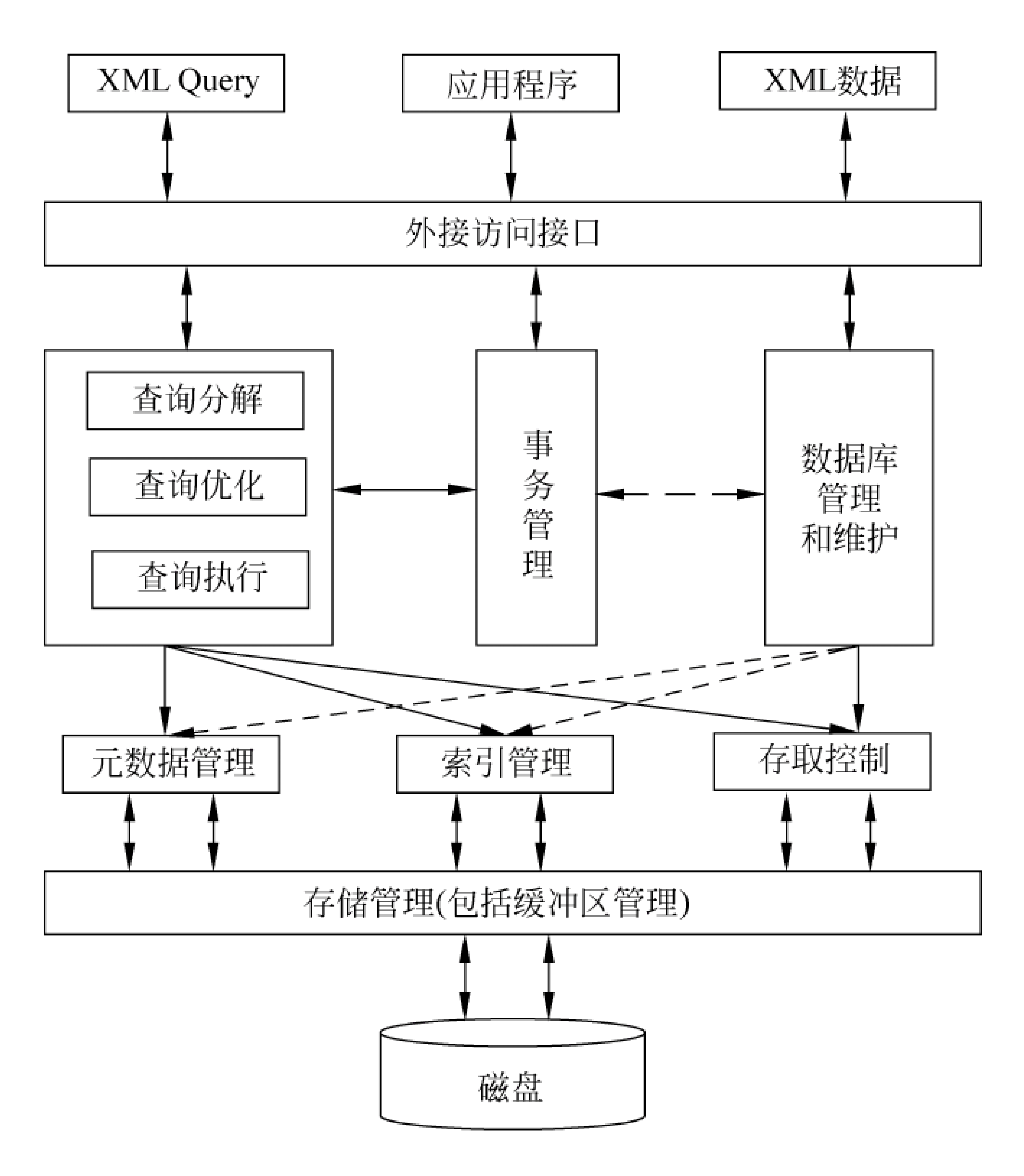


图 2-14 纯 XML 数据库系统的体系结构

不同的纯 XML 数据库系统的体系结构是不同的,但至少应该包含存储管理、索引管理、查询处理以及数据导入等模块,而一个完备的纯 XML 数据库还应该包含事务管理、并发控制、恢复技术等,并且应该提供良好的更新机制,此外,还应有相应的应用程序编程接口和友好的用户使用界面。

一个基于关系数据库的 XML 使能数据库系统体系结构如图 2-15 所示。

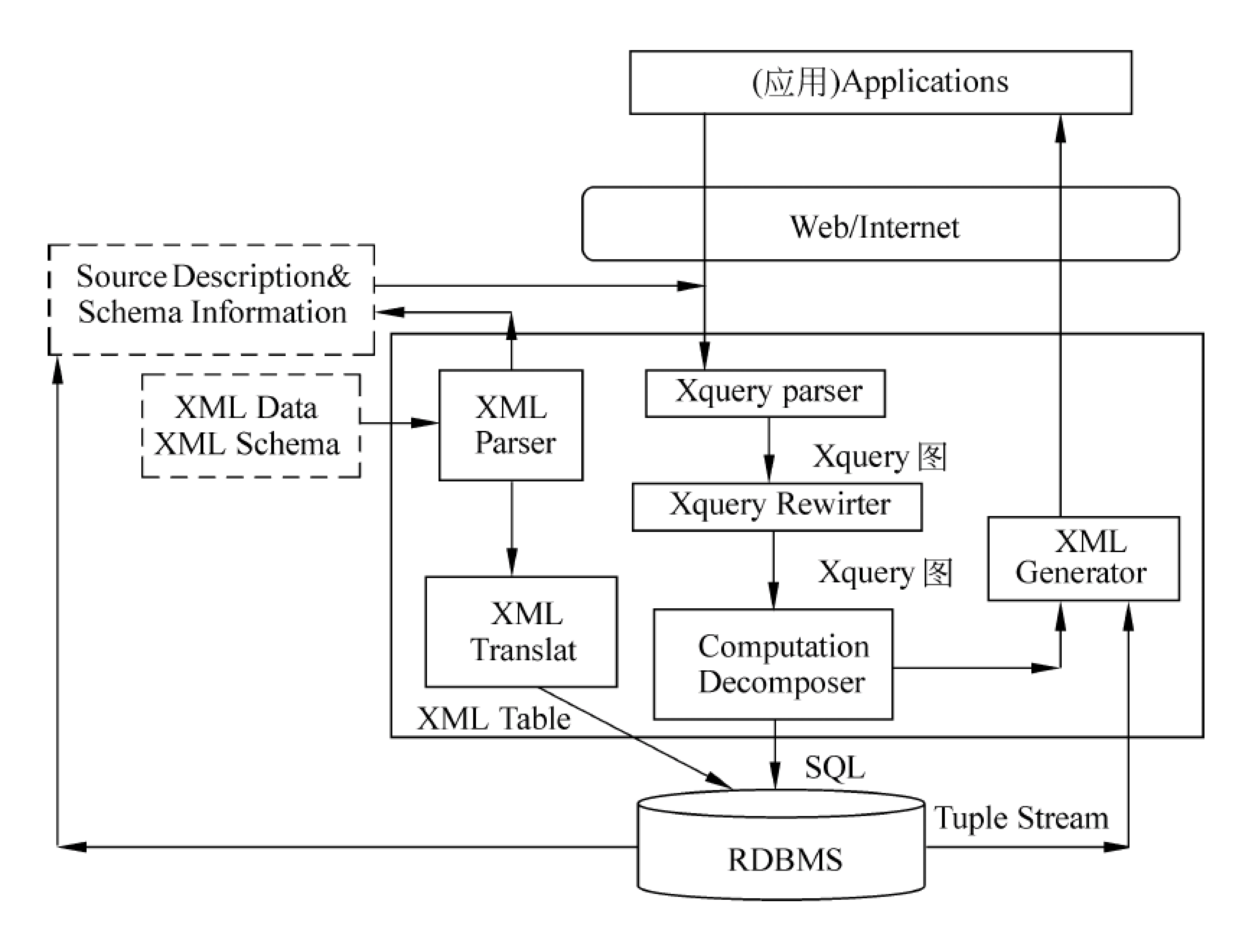


图 2-15　基于关系数据库的 XML 使能数据库体系结构

从图 2-15 中可以看出 XML 使能数据库可以充分利用关系数据库成熟的数据管理和查询及事务管理技术，只是外加 XML 数据管理的包装(wrapper)即可。将 XML 数据存储在关系数据库中会给数据管理和查询处理带来好处，关系数据库管理系统提供了强大的查询和修改数据的能力，使得可以利用在过去的版本上所进行的投资，就如同基于成本的优化和数据存储领域一样。例如，关系数据库中的索引技术已经广为人知，而且已经扩展到用于索引 XML 数据，这样就可以使用基于成本的决策来优化查询。而纯 XML 数据库系统必须提供相应的数据管理、索引管理及事务管理功能，比使能数据库系统复杂得多。

2. 数据存储策略的不同

XML 文档本质是序列化数据。序列化数据通常采用平面文件的形式，即将每一个 XML 文档分别存储在一个文本文件里，并且实现一个查询引擎，当查询被执行的时候，XML 文件被解析成驻留在内存的一棵树。只要查询计算还需要树中的节点，这棵树就必须驻留在内存里。一般来说，解析

的时间左右着查询计算的时间，但是这种方法慢得令人无法接受。为了提高这种方法的可用性，必须建立以下索引：利用 XML 元素在文本文件中的偏移量作为 ID，建立从标记 tag(Parent_off-set，Tag)映射到子偏移量 Child_offset 的路径索引以及从标记 tag(Child_offset，Tag)映射到父偏移量 Parent_offset 的反向路径索引，这两个索引非常有利于遍历 XML 文档或在 XML 文档中进行导航。而其他的索引建立从标记值(Tagname，Value)或属性值(Attribute_name，Attribute_value)到元素偏移量 Element_offset 的映射，这些索引有助于计算查询中的选择条件(谓词)。查询引擎可以利用这些索引检索与查询相关的 XML 文件的片段，极大地减少了解析时间。平面文件数据库最大优点就是容易实现，而且不需要使用数据库系统和存储管理器。但是，利用平面文件存储 XML 文档也有以下缺点：

(1) 每次访问 XML 文档时都需要解析它；

(2) 在查询处理期间，整个被解析的文件都必须驻留在内存里；

(3) 为了部分解决上述两个问题，可以在 XML 文档上建立外部索引。但当 XML 文档被更新时，索引是难以维护的。

除了平面文件以外，还可以将 XML 文档存储在传统的关系数据库系统里。最近的研究成果说明了如何将 XML 文档映射并存储到关系数据库系统中。这种方法的缺点是当前关系数据库系统与 XML 的负载不协调，而且通过诸如 SQL 的接口访问 XML 数据会招致与存储无关的额外负担。

利用关系的 DTD 方法。如果描述 XML 文档模式的 DTD 图中包含环，则必须用一个独立的表来打破这个环。在这种情况下，XML 数据库的索引可以完全建立在上述各个关系表上。

利用关系的边的方法。在 Edge 表上运用簇集策略对查询的性能也会有重要影响，一般都会选择按照 Edge 表上的 Tag 字段进行簇集，使得具有相同 Tag 的元素存储在一起。当然也可以选择按照“源结点”字段进行簇集，这种策略的好处是一个 XML 元素的所有子元素都存储在一起，因此重构原始的 XML 文档将非常快。但它的缺点是具有相同标记名的元素没有被簇集，因此对于这样的查询“把专业是计算机科学的所有学生都选择出

来”将会导致大量的随机 I/O。而建立在 Tag 字段上的簇集则表现出了较好的性能,但它对 XML 文档的重构没有任何好处。

另外一种所谓的属性方法。它是按 Edge 表的 Tag 字段对边表进行水平划分,不同标记(指 Tag 字段的值)的元组存储在各自不同的表里。这种方法以牺牲 Edge 表非常重要的属性 Tag 为代价而节省了存储空间。对属性方法来说,查询处理器需要 DTD 来决定哪些表包含子元素,因为子元素的标记没有存储在表中。需要注意的是,对于有很多 XML 文档的一个大集,属性方法可能导致大量的表。

对象管理器方法。在对象管理器里存储 XML 文档的明显方法就是把每一个 XML 元素存储成一个独立的对象,但是由于 XML 元素通常都非常小,因此这种方法的空间开销高得令人不敢问津。取而代之,把 XML 文档的所有元素存储在一个单独的对象里,而 XML 元素本身就变成了这个对象里的轻量级对象。同时,文献[2]用 1w_object,即记录来表示轻量级对象,而用 file_ob-ject 表示代表整个 XML 文档的对象。轻量级对象(1w_obiect)在文件对象(hle_object)中的偏移量(Offset)被用作该对象的标志(1w_oid)。

平面文件是存储 XML 文档的最简单的机制,但一般不支持索引查询,也不容易修改文档。关系型或面向对象数据库按照一定的粒度来存储 XML 文档,这使得对 XML 文档的访问比较容易也比较灵活,同时提高了查询和修改 XML 文档的效率,而且还可以方便地建立和维护各种索引。

文献[37]通过实验数据比较了这几种方法的性能。实验使用两个数据集:第一个数据集包含 250 个 XML 文档,总共 114MB[38];第二个数据集是 ODP,总共 140MB[39]。实验所用的关系数据库为 DB2V7.1,对象方法的底层利用系统 Shoret[40]来实现;DB2 和 Shore 都配置使用 30MB 的内存缓冲池。文本方法中没有缓冲池,其查询处理器使用所有可用的 256MB 物理内存;文本方法中的索引利用 BerkeleyDB(参阅 BerkeleyDBToolkit,http://www.Sleep-ycat.com/)来实现。在利用关系数据库的 DTD,Edge 和属性方法中,将用 XQuery 表达的查询手工翻译成 DB2 可执行的 SQL 查询

语句。

通过对存储比、XML 文档的重构和查询速度等指标的测试，初步的实验结果表明要想获得好的性能和紧凑的数据表示，相关 XML 文档的 DTD 信息至关重要。当 DTD 信息可利用时，DTD 方法的数据表示就很紧凑，而且该方法对不同的数据集和不同的查询均表现出优秀的性能。而利用适当的索引，文本方法可以获得与对象管理器方法相似的性能。但是，维护索引的代价将使这种方法只适用于 XML 文档更新不频繁的时候。

3. 查询处理的不同

早期的 XML 数据以文档(文本文件)的方式存储，以关键字查询等信息检索手段进行查询，简单易用，适合 XML 文档不频繁更新的场合。由于缺乏系统的存储和查询机制的支持，造成系统查询能力低，不能满足复杂条件的查询，更谈不上查询优化。一些现有的商业数据库管理系统扩充了处理 XML 数据的功能，利用现有数据库成熟的技术，把 XML 查询要求转换为数据库的查询表达，如 SQL，由数据库的查询引擎优化查询表达，产生查询执行计划并执行，最后再将查询的结果转换为 XML 数据。这种方法在一定程度上解决了查询复杂性的要求，但是多次转换带来的问题是效率的降低和查询语义的混淆，即阻抗失配的问题，主要体现在以下三个方面：

(1) 从本质上讲，传统的 RDBMS 不支持层次的和半结构化的数据形式，只有经过转换处理才能把嵌套的 XML 数据放到简单的关系表中。XML 是具有动态结构的数据，这正是 XML 可扩展性的关键，而传统的 RDBMS 不能处理这种数据。

(2) SQL 适合于查询受确定模式支配的表，它不是为具有动态、复杂特性的 XML 设计的。

(3) 传统的 RDBMS 查询引擎需要把针对 XML 的查询翻译成很多关系表的检索和连接运算，这不仅造成查询性能的下降，而且往往不能产生代价较小的查询执行计划。

NativeXML 数据库管理系统(纯 XML 数据库管理系统，NXDBMS)以

自然的方式处理 XML 数据，没有因数据模型转换而带来信息丢失和性能下降。NXDBMS 与非 NXDBMS 的区别在于：①有效地支持 XML 数据的自描述性、半结构化和有序性；②系统直接存储 XML 数据，而不是把 XML 数据转换成关系模型或者面向对象模型，由关系数据库或面向对象数据库存储；③直接支持 XML 查询语言，如 XQuery、XPath，而不是转换成 SQL 或 OQL（Object Query Language，对象查询语言）。

4. 事务管理的不同

事务管理的目的是为了保持事务的四个特性，即原子性、一致性、隔离性和持久性。成熟的事务管理技术主要是两个方面：一方面是恢复技术，它能够保证事务的原子性和持久性，并能够和人（程序员）保证一致性；另一方面是并发控制技术，它能够保证事务的孤立性，而并发控制技术则主要涉及锁的技术、时间戳技术等。

恢复技术的基石是日志。日志技术首先与 XML 数据库的存储粒度有关，如按子树存储，一个日志记录可能比较复杂，因为很难完全存储一个记录的前映像或后映像，这时的前后映像可能都比较大。当然也可以记录相应的操作和值，这时可能也会使一个日志记录较大，不过这样在恢复的时候可能节省磁盘 I/O。如果存储粒度较小，按结点存储，则可能会导致日志太“琐碎”。

再来看并发控制技术。在关系数据库里面，并发控制主要是通过锁机制实现的，这些锁机制大部分都可以用到 XML 中。最常见的封锁协议是两阶段封锁协议，它要求每个事务分两个阶段加锁和解锁。这在关系数据库中是可以的，因为在 SQL 中要访问的表是确定的。但是在 XML 数据库中，操作对象可能是未知的，典型的情况就是“跳转”（比如含 IDREF 属性的情况）。还有一种并发控制方法是多版本机制。锁机制里保证事务的可串行性的方法是当访问冲突时要么延迟一项操作，要么中止一项操作。比如 Read（X）操作，如果 X 的值已经被覆盖，则操作会被拒绝。多版本机制就是给每个更新的数据项保持多个版本，不同的事务可以访问不同的版本，从而

提高并发度。如上例中,如果给 X 保持新旧两个副本,则 Read(X)就不用被拒绝了。多版本机制主要还是基于时间戳,它能够明显提高并发度和性能。这种思想应该可以用到 XML 数据库中,尤其是像叶子结点访问比较频繁,用多版本可以缓解频繁访问的压力。当然,如何控制,如何选择正确的版本可能是比较复杂的问题。

在现实当中,以数据为中心和以文档为中心的文档之间的差别不一定很明显。例如,以数据为中心的文档,比如发票,可能含有大粒度的、结构不规则的数据比如零件说明;以文档文中心的文件,如用户手册,可能包含细粒度的结构规则的数据(通常为元数据),比如作者和修订日期。其他例子包括法律和医学文书,虽然以松散的形式写成,但是却包含离散的数据,例如日期、名称和操作程序,出于法规的原因通常要以完整的文件形式存储。

弄清文件的这两种特点有助于选择数据库的类型。一般来说,将数据存储于传统的数据库,例如关系型,面向对象型或层次型数据库,这可由第三方的中间件完成或由数据库本身提供内在支持。对于后者,该数据库被称作支持 XML 的(XML-enabled)。文档可被存储在原生(native)XML 数据库或内容管理系统(建在原生 XML 数据库之上专门用来管理文档的程序)中。

这些原则并不是绝对的。如果对 XML 特有的功能不很看重,数据,特别是半结构化的数据可以存储在原生 XML 数据库中,文档也可以存储到传统数据库。何况传统数据库与原生 XML 数据库之间的界限越来越模糊,传统数据库增加了原生 XML 的能力,而原生 XML 数据库也增加了对文档存储在外部(通常为关系型)数据库的支持。

参考文献

[1] (美)Mark Graves 著. XML 数据库设计. 尹志军等译. 北京:机械工业出版社,2002,13～15
[2] XML Database [EB/OL]. http://www. Xmldb. org/faqs. html
[3] What it is ? [EB/OL]. http://www. ozone-db. org/frames/home/what. html

[4] Tian Feng, DeWittD J, et al. The Design and Performance Evaluation of Alternative XML Storage Strategies. SIGMOD Record, March 2002, 31(1): 68～79

[5] Abiteboul S, Cluet S, et al. Queuing and updating the file. In: Proc. of 19th International Conference on Very Large ta Bases, Dublin, Ireland 1993. 54～66

[6] Floreseu D,KossmanD. Storing and Querying XML Data using an RDBMS. IEEE Data Engineering Bulletin, September 1999, 22(3): 27～34

[7] Kanne C, Moerkotte G. Efficient storage of XML data. In: Proc. of the 16th International Conference on Data Engineering, 28 February-3 March, 2000, San Diego,California, USA IEEE Computer Society 2000. 198～205

[8] J. Shanmugasundaram, K. Relational Databases for Querying XML Documents: Limitations and Opportunities, In: Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, UK 1999. 302～314

[9] Bourret R. XML and Database [EB/OL]. http://www. rpbourret. com/xml/XMLAndDatabase. htm

[10] Bourret R. XML Data Products: Native XML Databases[EB/OL]. http://www. rpbourret. com/xml/ProdsNative. htm

[11] Kanne C C and Moerkotte G. Efficient Storage of XML Data. In Proceedings of 16th ICDE, San Diego, California, USA, February 2000. 198～210

[12] Mchugh J, Abiteboul S, Goldman R, et al. Lore: A Database Management System for Semistructured Data. ACM SIGMOD Record, 1997, 26(3): 54-66

[13] Jagadish H V, AL-Khalifa S, et al. TIMBER: A Native XML Database. Technical Report, University of Michigan, April 2002

[14] Kanne C, Moerkotte G. Efficient storage of XML data. In: Proceedings of the 16th International Conference on Data Engineering, 28 February-3 March, 2000, San Diego,California,USA IEEE Computer Society 2000. 198～210

[15] 罗道峰,孟小峰,安靖. OrientStore: Native XML 存储方法. 第 20 届全国数据库学术会议论文集. 重庆: 计算机科学,2003. 105～110

[16] Meng Xiaofeng, Luo Daofeng, Lee Mong Li, et al. OrientStore: A Schema Based Native XML Storage System. In: Heuer A et al Eds. Proceedings of the 29th VLDB International Conference On Very Large Database. Berlin, Germany. September 9-12, 2003. San Francisco: Morgan Kaufmann Publishers, 2003. 1057～1060

[17] 孟小峰,王宇,罗道峰等. OrientX: 一个 Native XML 数据库系统的实现策略. 第 20 届全国数据库学术会议论文集. 重庆: 计算机科学,2003. 111～115

[18] Fiebig T, Helmer S, Kanne C C, et al. Anatomy of a Native XML Base Management System. The VLDB Journal, 2003, 11(4): 292～314

[19] Goldman R and Widom J. DataGuide: Enabling Query Formulation and

Optimization in Semistructured Databases. In: Jarke M et al Eds. Proceedings of the VLDB International Conference on Very Large Databases. Athens, Greece. August 25-29, 1997. San Francisco: Morgan Kaufmann Publishers, 1997. 436～445

[20] The Index Fabric: a Technical Overview. Technical Report, February, 2001. http://www.rightorder.com/technology/overview.pdf

[21] Cooper B F, Sample N, Franklin M J, et al. A Fast Index for Semistructured Data. In: Apers P M G et al Eds. Proceeding of the 27th VLDB International Conference on Very Large Database. Rome, Italy. September 11-14, 2001. San Francisco: Morgan Kaufmann Publishers, 2001. 341～350

[22] Kha D D, Yoshikawa M, Uemura S. An XML Indexing Structure with relative region coordinate. In: Reuter A et al Eds. Proceeding of the 17th IEEE ICDE International Conference on Data Engineering. Heidelberg, Germany. April 2-6, 2001. Los Alamitos: IEEE Computer Society, 2001. 313～320

[23] McHugh J, Widom J, Abiteboul S, et al. Indexing Semistructured Data. Technical Report, January 1998. http://www.db.stsnford.edu/lore/pubs

[24] Fiebig T and Moerkotee G. Evaluating Queries on Structure with Extended Access Support Relations. In: Proceedings of the WebDB International Workshop on the Web and Databases(Informal Proceedings), Dallas, Texas, 2000. 41～46

[25] 万常选,刘云生. 基于关系数据库的 XML 数据管理. 计算机科学,2003,30(8): 64～68

[26] 王淑芬,朱晓亮等. 基于模式约束的 XML 数据存储和索引技术[J],数学的实践与认识,2010,40(14): 72-79

[27] Shamugasundaram J, Shekita E J, Barr R, et al. Efficiently Publishing Relational Data as XML Documents. In: Abbbad A E et al Eds. Procedings of the 26th VLDB International Conference on Very Large Database. Cairo, Egypt. September 10-14, 2000. San Francisco: Morgan Ksufmann Publishers, 2000. 65～76

[28] Carey M, Kiernan J, Shamugasundaram J, et al. XPERANTO: A Middleware for Publishing object-relational data as XML Documents. In: Abbbad A E et al Eds. Procedings of the 26th VLDB International Conference on Very Large Database. Cairo, Egypt. September 10-14, 2000. San Francisco: Morgan Ksufmann Publishers, 2000. 646～648

[29] Shamugasundaram J, Kiernan J, Shekita E J, et al. Querying XML Views of Relational Data. In: Apers P M G et al Eds. Procedings of the 27th VLDB International Conference on Very Large Database. Rome, Italy. September 11-14, 2001. San Francisco: Morgan Kaufmann Publishers, 2001. 261～270

[30] Fernandez M, Tan W, Suciu D. SilkRoute: Trading Between Relations and XML. In: Herman I et al Eds. Procedings of the 9th International World Wide Web

Conference（WWW'00）. Amsterdam，The Netherlands. May 15-19，2000. Amsterdam：Foretec Seminars Inc，2000. 723～725

[31] Fernandez M，Tan W，Suciu D. Publishing Relational data as XML：The SilkRoute Approach. IEEE Data Engineering Bulletin，2001，24(2)：12～19

[32] Quanzhong Li，Bongki Moon，Indexing and Querying XML Data for Regular Path Expression[C]. Proceedings of the 27th International Conference on Very Large Database，Roma，Italy，September 11-14，2001. San Francisco：Morgan Kaufmann Publishers，2001：361-370

[33] Yoshikawa M，Amagara T，Shimura T，et al. XREL：A Path Based Approach to Storage and Retrieval of XML documents using Relational Dtabases. ACM Transactions on Internet Technology(TOIT). 2001，1(1)：110～141

[34] Jiang Haifeng，Lu Hongjun，Wang Wei，at el. Xparent：An Efficient RDBMS-based XML Database System. In：Hiong Ngu A H A et al Eds. Proceding of the 18th IEEE ICDE International Conference on Data Engineering. San Jose，California，USA. February 26～March 1，2002. Los Alamitos：IEEE Computer Society，2002. 335～346

[35] Shamugasundaram J，Tufle K，Zhang C，et al. Relational Databases for Querying XML Documents：Limitations and Opportunities. In：Atkinson M P et al Eds. Procedings of the 25th VLDB International Conference on Very Large Database. Edinburgh，Scotland. September 7-10，1999. San Francisco：Morgan Kaufmann Publishers，1999. 302～314

[36] 万常远，刘云生，徐升华等. 基于区间编码的 XML 索引结构有效实现结构连接. 计算机学报，2005，28(1) 113～127

[37] 门爱华，冯建华，周立柱. XML 数据库存储策略综述. 计算机科学. 2005，32(9)：13～18

[38] Carey M，De Witt D，et al. The BUCKY Object-Relational Benchmark. In：Proc. of ACM SIGMOD Intl. Conf. on Management of Data，May 13-15，1997，Tucson，Arizona，USA SIGMOD Record，Ju ne 1997，2(12)：273～284

[39] Open Directory Project，http://www. dmoz. org/

[40] Carey M，De Witt D，et al. Shoring Up Persistent Applications. In：Pro c. of ACM SIGMOD Intl. Conf. on Management of Data，May 24-27，1994，Minneapolis，Minnesota，USA SIGMOD Record，June 1994，23(2)：383～394

第3章　基于关系数据库的 XML 文档管理

设计数据库时，要分析数据库必须存储的信息及这些信息组成部分之间的关系。通常数据库的结构为数据库模式（Database Schema），传统的关系数据库采用“实体/联系”（E/R）模型来描述数据关系，并依此进行数据库设计。XML 文档是半结构化数据，E/R 图不能完全体现数据之间的关系，所以首要解决的是 XML 数据库的数据模型问题。

3.1　XML 数据模型

数据模型是 XML 数据管理研究领域的核心问题之一，用来给出 XML 数据以及数据上操作的精确语义，是 XML 数据的查询处理和优化的基础。部分研究者利用传统数据模型（如关系模型和面向对象数据模型）来表示 XML 数据的结构和语义。其中，Stanford 的 R. Goldman 等人利用 OEM 模型来表达 XML 数据的结构和语义[1]，研究者也开展了大量基于关系模型的 XML 数据管理的研究工作[2-7]。这些方法的最大问题是：XML 数据上的一个操作需要用这些模型上的一系列操作来表示，因此显得力不从心。

为此，研究者给出了两个 XML 模型——树模型和图模型。这两种模型都是直观上的模型，不是严格意义上的数据模型。因此，提出一种能够有效表达 XML 数据的模型以及给出该模型上 XML 数据操作的形式化定义变得尤为重要。如何利用数学的方法严格描述 XML 数据以及数据上的操作，进而完成 XML 数据查询的代数优化，成为 XML 数据管理领域一个十分重要的研究问题。

在 XML 数据模型和代数操作方面，现已取得了一些研究成果：

（1）研究者用四种模型对 XML 数据进行描述：关系模型、面向对象模型、树模型和图模型。

面向对象模型可以方便地表达出 XML 数据的结构以及语义，但为了支持路径表达式查询，操作必须是面向过程的，需要复杂的数据导航，并不适应 XML 数据管理的需要；另外，由于 XML 数据半结构化的特点，为存储 XML 数据，将会产生大量磁盘碎片。如果用关系模型或嵌套关系模型来描述 XML 数据的结构和语义，其操作需要用大量昂贵的 join 操作来表示[4]，并不适应 XML 数据管理的需要；此外，用关系和面向对象模型表示 XML 数据的结构和语义，将 XML 数据查询操作的语义映射到相应模型的开销也很大。

研究者根据 XML 数据的结构和语义，采用了两种直观的 XML 模型——树模型和图模型。其中树模型得到了广泛的应用，它把一个 XML 文档视为一棵树，把多个 XML 文档看作一个森林。因为树模型可以很容易地表达出 XML 数据的层次结构，在 XML 数据管理的研究中，人们自然地把它视为 XML 的直观数据模型。但它忽略了 XML 数据的一些特性，如 XML 数据的模式信息以及元素结点之间的引用关系。图模型把 XML 数据视为一个有向图。直观上，图模型可以表达单个 XML 文档的语义，但是，目前还没有研究给出图模型的精确定义。树模型和图模型都是对 XML 文档进行建模，对于具有相同模式而具体数据不同的 XML 文档，这两种模型在表达上是不同的，这并不适合数据管理的需要。为此，M. Arenas 等人利用正则表达式给出了 XML 模式和数据的精确定义[8]，但该研究并未进一步给出 XML 数据上操作的定义。

（2）在 XML 代数操作的研究方面，研究者提出了一些 XML 的代数操作。

在 Christophides 以及 Ludascher 等人关于中间件的研究中，给出了一些适合中间件数据交换的 XML 操作代数[9-10]。这些研究的最主要特点是代数操作的输入和输出都是元组，这些研究是早期的 XML 数据操作代数。在

W3C 标准中给出 XML 路径表达式的查询语言 XQuery,它的输入和输出都是 XML 文档元素的列表。此外,研究者还提出了其他一些 XML 代数:Buneman 等人提出的一种半结构化数据的查询语言和代数[11];Beeri 等人提出的 SAL[12];Jagadish 等人提出的 TAX[13];Fernandez 等人于 2001 年也给出相应的代数操作[14];孟小峰等人提出了 OreintXA[15]。但这些研究并没有给出 XML 代数操作的精确语义。

由于已有的 XML 模型在以下几方面各有不足之处:

- 无法表达 XML 数据的复杂语义;
- 没有给出完整的代数操作的定义;
- 在模型上没有给出数据修改操作的明确定义。

目前,大多数研究直接采用半结构化数据模型,其中最有代表性的是 OEM 模型[4]以及 XQuery1.0 和 XPath 2.0 数据模型[5]。本章将对 XML 数据模型进行探讨,通过对 XQuery 数据模型的扩展,提出一种基于关系数据库的存储策略[16],并研究实现了基于扩展 XQuery 的更新语言 XUL[17]。

3.1.1 对象交换模型

对象交换模型(Object Exchange Model,简称 OEM)是专门为了表示半结构化数据而提出的。该模型产生于 1997 年,应用在斯坦福大学 Abiteboul 等人开发的半结构化数据库管理系统 Lore(light weight object repository)[1]中。在 OEM 模型中,所有实体都是对象(object),一个 OEM 对象是一个四元组:(label,oid,type,value)。其中,label 是对象标记,表示对象的名称,标记有两个作用,用于区分不同的对象、表达对象的含义;oid 是对象标识符,不同对象的 oid 是唯一的;type 表示对象类型,OEM 定义了两种类型的对象:复杂类型、原子类型;value 是对象的取值。当 type 是原子类型时,称对象为原子对象(atomic object),此时 value 是该类型的一个原子值,如 integer、real、string、gif 等;当 type 是复杂类型时,对象称为复杂对象(complex object),此时 value 是对象标识的集合(或列表),即复杂对象由若干子对象构成,子对象可以是复杂对象或者原子对象。

XML 推出后,OEM 数据模型又专门进行了基于 XML 的扩展,在基于 XML 的 OEM 数据模型中,一个 XML 元素用一个二元组(eid,value)表示。eid 是该元素唯一的标识符;value 是元素的值,该值可以是一个文本串形式的原子值,也可以是包含以下 4 种成分的复杂值:

(1) 用字符串表示的该元素标记;

(2) 属性名——原子值对列表,其中属性名是一个字符串,代表属性的名称,原子值是该属性的取值,原子值的类型可以是 integer、real、string、ID、IDREF、或者 IDREFS 等;

(3) 一个有序的形如(label,eid)的链接子元素列表,其中 label 是一个字符串,表示子元素的标记,eid 为链接元素的标识符,链接子元素通过 IDREF 或者 IDREFS 属性类型加以说明;

(4) 一个有序的形如(label,eid)的一般子元素列表,表示非链接子元素,其中 label 是子元素的标记,eid 为子元素的标识符。

对一般子元素与链接子元素加以区分是为了使模型可以同时支持文法模式和语义模式。OEM 数据模型可以用有向边标记图模型表示,称为 OEM 图模型。在 OEM 图模型中,结点代表对象或者原子值,边对应元素名称。图 3-1 显示的是一个简单的 XML 文档和它对应的 OEM 图模型。从图中可以直观地看出所描述数据的内容和结构特征:元素标识符(eid)以 &1、&2……的形式出现在结点中;指向该结点的边上的标记表示结点的名称(元素名称);元素值以及属性名称和对应的属性值显示在相应结点的旁边。图中元素之间以及元素与对应的原子值之间的边用实线表示,IDREF 属性用斜体表示,交叉链接边用虚线表示,子元素的顺序按照自左至右排列。

3.1.2　XQuery 数据模型

Infoset 模型、XPath 数据模型、DOM 模型、XQuery 1.0 和 XPath 2.0 数据模型都是 W3C 推荐的 XML 数据模型,这些模型都是 OEM 的变体。其中,XQuery 1.0 和 XPath 2.0 数据模型[18]是 W3C 的候选推荐标准,该模型来源于 Infoset 模型,目前由 XSLT 2.0、XQuery 1.0 和 XPath 2.0 三部分

```
<DBGroup>
    <Member Name=  "smit h" Advison="m1 ">
        <Age>28</Age>
    </ Member >
    < Member ID= "m1 " Progect= "p1 ">
        <Name>Jones</ Name>
        <Advisor>Ullman</Advisor>
    </ Member>
    <Project ID="p1 " Member=  "m1 ">
       <Title>Lore</Title>
    </ Project>
</ DBGroup>
```

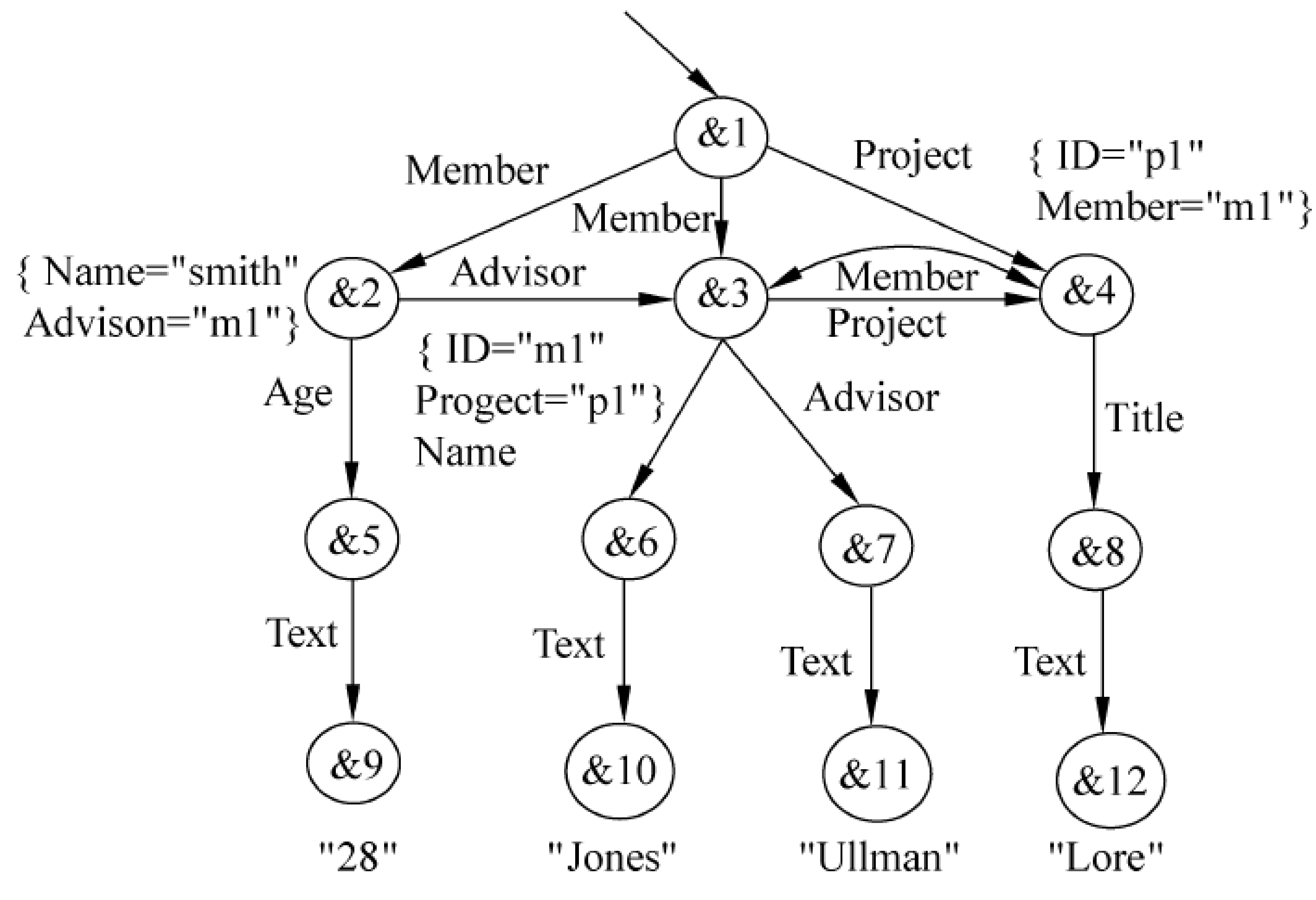


图 3-1　一个 XML 文档及其对应的 OEM 图模型

组成。模型定义了 XML 文档的表示形式，描述了实现查询必须理解的数据项和形式语义的基础，对 XSLT 和 XQuery 处理器所能够处理的信息进行了说明，为 XSLT、XQuery、XPath 语言中的表达式所允许的值进行了定义。W3C 是这样定义这个数据模型的："XQuery1.0 and XPath2.0 数据模型可以达到两个目的：第一，它定义了输入到 XPath、XSLT 和 XQuery 处理器的信息；第二，它定义了 XPath、XSLT 和 XQuery 中所允许的表达式的值。如果一个语言中的每个表达式的值都保证在数据模型中，那么就说这个语言关于这个模型是封闭的。XSLT2.0、XQuery1.0 和 XPath2.0 关于这个模型都是封闭的。"

在该模型中，把一个 XML 文档看成是由若干(数据)项(item)组成。一个 XML 文档可以表示成一棵树(XML 文档树)，文档中所有的 item 在 XML 文档树中用结点表示(node)，一个 item 是树的一个结点。模型定义了 7 种结点类型：文档结点、元素结点、属性结点、名称空间结点、处理指令结点、注释结点、文本结点。

(1) 文档结点

文档结点为树的根结点，代表整个文档集合，除此之外，文档结点不再出现。XML 文档的根元素结点是文档结点的孩子。出现在 XML 文档的序言中(即根元素之前)和根元素之后的处理指令及注释直接作为文档结点的孩子结点(处理指令结点及注释结点)。

(2) 元素结点

XML 文档中的每个元素都对应于 XML 文档树中的一个元素结点。元素结点的孩子可以是元素结点、注释结点、处理指令结点、属性结点、命名空间结点及其内容的文本结点。对内部及外部实体引用将被扩展，字符引用也被分解。

(3) 属性结点

每个元素结点可以有相关联的属性结点集，元素结点是它的每个属性结点的父结点。XML 标准详细说明了属性结点的字符串，它是一个规范化的值。

(4) 命名空间结点

每个元素结点可以有相关联的命名空间结点集，它是在 XML 文档中声明的在元素的范围内有效的每个命名空间前缀(包括 XML 前缀，它由 XML 命名空间建议隐式地声明)，以及默认命名空间。元素结点是它的每一个命名空间结点的父结点。命名空间结点的字符串是命名空间前缀所绑定的命名空间 URI，如果它是相对的，则必须被解析。

(5) 处理指令结点

除了在文档类型声明中出现的处理指令外，每个处理指令对应一个处理指令结点。处理指令结点的字串值是处理指令的内容，不包括起始的

“＜?”和“?＞”。

（6）注释结点

除了在文档类型声明中出现的注释外，每条注释对应一个注释结点。注释结点的字串值是注释内容，不包括起始的“＜!”和“!＞”。

（7）文本结点

文本结点由字符数据组成。每个文本结点都包含尽可能多的字符数据，而且不会出现两个相邻的文本结点。文本结点的字串值是其字符数据。

一个数据模型的实例包含一个或多个 XML 文档或 XML 文档片断，每个都表示为由若干结点构成的树。一个 XML 文档对应的树的根结点是一个文档结点。在这个数据模型中，每个结点具有两种类型的值，分别是其字串值和类型值。字串值就是该结点用字符串形式表现的值；而类型值则是结点声明的类型的值。元素（结点）和属性（结点）的取值为原子类型（atomic type）值域中的值，包括 XML Schema 和 xdt：untypedAtomic 定义的所有类型。每个结点有一个唯一的结点标识（node identity）。另外，元素结点和属性结点还可以有类型值（typed values）、字符串值（string values）和名称，这些内容可以从结点中抽取出来（参见图 3-5 和图 3-10）。

3.2　基于 Schema 约束的 XML 文档存储和索引技术

可扩展标记语言 XML 作为一种与平台无关的数据表示形式已被广泛采用，特别是电子商务、Web 服务等应用的普及发展，使得 XML 类型的数据成为当前主流的数据形式，数据交换已成为 XML 技术的主要驱动力之一。

XML 数据的基本形式是 XML 文档，XML 数据库就是一个 XML 文档的集合，这些文档是持久的并且可操作的。因为低层的存储表达对上层的查询处理和优化有着重要的性能影响[19]，所以如何以最好的方式存储 XML 文档已经成为研究热点。关系数据库以其强大的查询功能和科学严密的数据库设计基础成为首选对象。曾任 ACM 数据管理专门组主席的计算机科学教授 Michel Stonebraker 将数据库的应用市场描述为一个 2 乘 2 的矩

阵[20](如图 3-2 所示),预言对象—关系数据库管理系统将是数据库市场的主流,并指出将对象—关系型引擎与关系型的存储管理器结合起来是一个好的策略。近年来,Oracle Database、IBM DB2 和 MS SQL Server 对 XML 数据的管理体系基本采用关系和对象—关系数据管理的技术支持,通过用户定义的函数和存储过程来实现对 XML 数据的存取、存储选择和变换。

	关系型 DBMS	对象-关系 DBMS
	文件系统	面向对象 DBMS
	简单数据	复杂数据

图 3-2　DBMS 分类矩阵

对于 XML 数据的约束,Brett Mclaughlin 曾这样阐述:一个有效的 XML 文档必须满足它引用的模式中设置的所有约束条件[21]。一个不是良构(well-formed)的或者无效(invalid)的文档容易引起歧义,因此引入 XML 数据模式(XML Schema,XML 模式)主要有三个目的:

(1) 模式可以约束表、属性或类层次,使用户理解数据库的结构,以便形成有意义的查询;

(2) 查询处理器必须依赖模式为计算查询结果而设计有效的查询计划[22];

(3) 如果不知道一个特定文档结点值的数据类型,就不可能准确知道在 Xpath 中比较操作符的含义。

XML Schema 是 W3C 推出的一个新的工作草案,弥补了 DTD 存在的问题以及局限性,为 XML 文档提供丰富的语法结构,支持复杂数据类型和文档片段,能够表示元素的复杂约束,并对类继承提供了广泛的支持。本节从符合 XML 模式的有效的 XML 文档数据角度出发,提出利用关系数据库来存储模式信息和文档数据的方法,该方法具有以下特点:

(1) 将文档结点信息和结构信息有机结合共同存储于关系表中,提高了复杂路径查询的导航效率;

(2) 强调模式约束文档,有利于保证 XML 文档的查询和更新的有效性;

(3) 有效利用关系数据库系统的索引技术和存储索引维护策略。

3.2.1 对现有 XML 数据存储管理技术的分析

根据第 2 章内容可知，XML 数据库的存储策略主要有四种：利用文件系统的平面文件、利用成熟的 RDBMS（Relational Database Management System，关系数据库管理系统）、利用对象管理器或 OODBMS（Object-Oriented Database Management System，面向对象数据库管理系统）、采用全新的 Native XML 数据库管理系统。实验结果表明关系数据库存储方法优于其他方法。

近几年所提出的 XML 到关系数据库的模式映射方法，主要分为两类：第一类方法映射产生的关系模式依赖于 XML 模式，在处理海量的、具有不同结构的 XML 数据时有很大的局限性，原因是修改模式定义带来的更新粒度较大。第二类映射方法产生的关系模式独立于 XML 模式，如文献[2]以一个三元关系存储 XML 文档。这种存储方法能够有效解决第一类映射方法中出现的部分问题，但对于 XML 路径查询，通常需要遍历整个数据库，进行多次连接操作才能确定路径的正确性并完成查询工作，因而效率相对较低。如何将两种技术有机结合，既能处理海量数据的存储问题，又能保证数据查询和更新的高效性成为我们研究的重点。

XML 数据库的索引技术对 XML 数据查询处理起着至关重要的作用，如果没有索引的支持，将带来很大的 I/O 代价和语义支持方面的限制。对于 XML 数据库索引技术的研究越来越受重视。例如 Stanford 大学的 Lore 系统中引入 DataGuides 用于保证数据库数据的一致性[23]，并建立了值索引、标签索引、边索引和路径索引。文献[4]提出了 5 种索引结构：Name Index、Element Index、Attribute Index、Structure Index、Value Table，检索方法是将复杂的 Xpath 查询路径表达式分解为简单路径，然后经过 EE（Element and Element Join Algorithm）、EA（Element and Attribute Join Algorithm）、KC（Kleene Closure Join Algorithm）的连接操作获得结果。XISS（XML Indexing and Storage System）索引兼备路径记录和结构索引、树接点编码两个特点，但索引结构中关于路径记录和节点编码的两部分

(Structure Index 和 Element Index)相互独立,造成索引的空间代价。

3.2.2　基于 Schema 约束的 XML 数据存储和索引

严格地讲,XML 模式并不是指哪一种模式语言,XML Schema 仅是多种选择中的一种。但是由于 XML Schema 的特殊地位以及与日俱增的影响力,XML Schema 有望成为主流的模式语言,本章中提及的模式就是 W3C 制定的 XML Schema。

一般来说,对于 XML 数据存储,可能有多个 XML Schema,而对于每个 XML Schema,可能对应多个 XML 文档。本文所提出的索引结构就是针对这样的数据存储。

本节将从检索效率和更新维护的角度出发,提出一种基于 Schema 约束的 XML 文档的模式和数据索引、存储方法,不仅改善存储效率和提高检索速度,更重要的是有利于数据更新操作。

1. XML 模式信息的存储和索引

文档实例与模式之间有多种对应关系,一个模式可以描述许多有效的文档,同样一个文档也可以被许多模式描述。模式最重要的处理是模式的验证,所以其信息必须单独存储。

定义 3-1　模式语义树[16]:对应于 XML Schema 定义的一棵有向树,根元素是 Schema,入度为 0,层次为 0;出度为 0 的结点称为叶结点,标记元素或属性的数据类型,用椭圆表示;入度和出度均不为 0 的结点称为分支结点,分为属性结点和元素结点,矩形框表示元素,三角形表示属性。矩形框两侧标识对元素或属性的限制,如最大最小值、允许空值、唯一性及是否主键等约束等;有向边标识父结点指向子结点,数字标识父结点的所有子结点的顺序(左右序)。图 3-4 是图 3-3 中 XML Schema 对应的模式语义树。

XML Schema 本身就是 XML 文档,一般不会太大,更新不会太多,所以采用深度优先(先根遍历)为每个结点编码,叶子结点不参与编码。建立模

```
<xs:schemaxmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Customer" type="CustomerType">
</xs:element>
<xs:complexType name="CustomerType">
  <xs:sequence>
    <xs:element name="CustomerID" type="xs:string" />
    <xs:element name="Order" type="OrderType" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="OrderType">
  <xs:sequence>
    <xs:element name="OrderID" type="xs:integer" />
    <xs:element name="LineItem" minOccurs="1" maxOccurs="unbound"
type="LineItemType" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="LineItemType">
  <xs:sequence>
    <xs:element name="ProductID" type="xs:integer" />
    <xs:attribute name="Quantity" type="xs:integer" />
    <xs: attribute name="UnitPrice" type="xs:decimal" />
  </xs:sequence>
</xs:complexType>
<xs :key name="KI">
    <xs: selector xpath="Customer"/>
    <xs: field xpath="CustomerID"/>
</xs: key>
</xs:schema>
```

图 3-3 一个 XML Schema 实例

式结构信息：

Schema_Structure(SchemaID, name, ea_flag, preorder, level, parent_preorder, sequence, type)
Schema_Constrain(SchemaID, name, min, max, restriction)

其中 Schema_Structure 是主表，用于存放模式结构信息。SchemaID 为 XML Schema 模式文档标识符，便于多模式的 XML 数据管理(如果数据库只有一个模式，可以省略)。ea_flag 为元素或属性标识，0 表示复合元素，1 表示属性，2 表示简单元素。Schema_Constrain 为辅表，用于存储模式中元素和属性的限制条件；max 和 min 表示数据的最大最小值；restriction 表示限制类型，如 key 表示关键字；scope 表示取值范围约束；sequence 表示

元素的子元素或属性有序等等。如图 3-4 中的 Order、LineItem 和 CustomerID 等元素，如果大部分元素有限制，可以将两个表合并为一个(将模式信息分别存储在两个关系表中，主要是考虑元素限制不多时，会产生大量空值)。

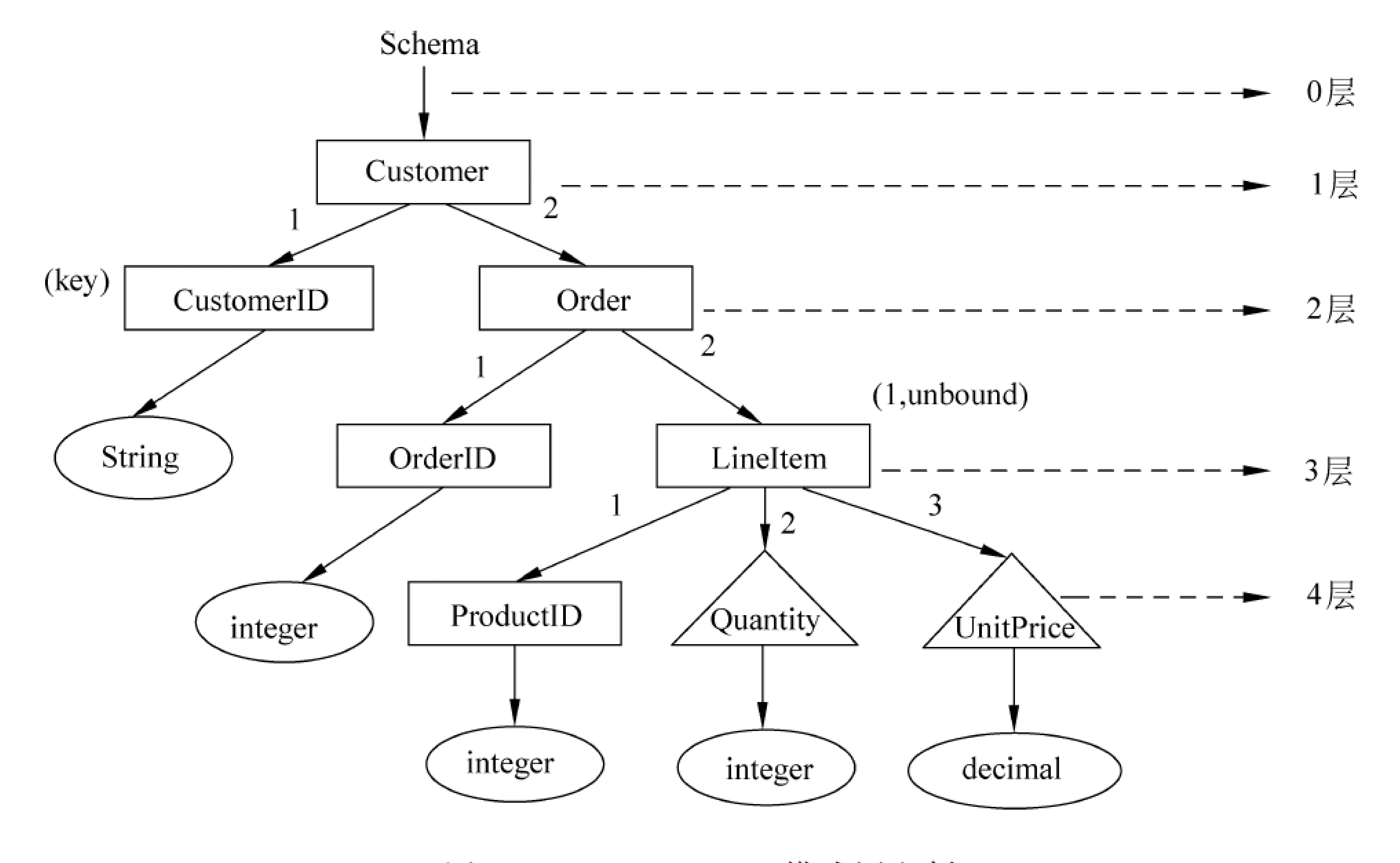


图 3-4　XML Schema 模式语义树

与图 3-4 对应的模式结构信息表和约束信息表如表 3-1 和表 3-2 所示。Schema_Structure 表主键为 preorder，以 SchemaID 为关键字建立 B^+-Tree 索引，对于相同 XML 模式，按照 preorder 字段建立辅索引，为路径查询实现快速导航。

表 3-1　模式结构信息表 Schema_Structure

SchemaID	name	ea_flag	preorder	level	parent_preorder	sequence	type
1	Customer	0	1	1	0	0	complex
1	CustomerID	2	2	2	1	1	string
1	Order	0	3	2	1	2	complex
1	OrderID	2	4	3	3	1	integer
1	LineItem	0	5	3	3	2	complex
1	ProductID	2	6	4	5	1	integer
1	Quantity	1	7	4	5	2	integer
1	UnitPrice	1	8	4	5	3	decimal

表 3-2 模式约束信息表 Schema_Constrain

SchemaID	name	min	max	restriction
1	CustomerID			key
1	LineItem	1	null	scope
1	LineItem			sequence
1	Order			sequence

2. 加载 XML 数据的有效性验证

验证用来判断文档是否符合模式中所描述的所有约束，涉及到检查文档实例中所有的元素和属性，从而确定它们是否有声明，以及它们是否符合这些声明及相应的数据类型定义。验证过程将检查以下内容：

- 数据的正确性。根据模式来验证并不能完全保证数据都是正确的，但是它可以显示无效的格式或者超出范围的值。
- 数据的完整性。检查是否所有必须的信息都存在。
- 对数据的共同理解。确保理解文档的方式与语法分析器相同。

有了 XML Schema 模式信息，就可以对 XML 文档数据进行有效性检验。如图 3-5 的文档树中，有两处与模式冲突，一处是模式中不存在“PO”元素，一处是“Quantity”和“UnitPrice”属性顺序颠倒。当然应该在加载数据的同时，即解析数据过程中实现检验和矫正过程，或修改 XML 文档或重新定义 XML 模式。

定义 3-2 文档树：对应 XML 文档数据的一棵有向树，符号含义与对应的语义模式树相同，只是叶子结点表示元素或属性的具体取值。

定义 3-3 有效 XML 文档：如果 XML 文档中的元素、属性的定义(包括名称、数据类型及约束定义)与对应模式中定义没有冲突，则认为此文档是有效的。有效 XML 文档树与对应语义模式树同构。

3. XML 文档数据的存储和索引

对有效 XML 文档树(如图 3-5 中删除 PO 元素，调换 UnitPrice 和 Quantity 属性)结点采用 Li-Moon 编码[25]方案，为每一个节点赋予一个二元

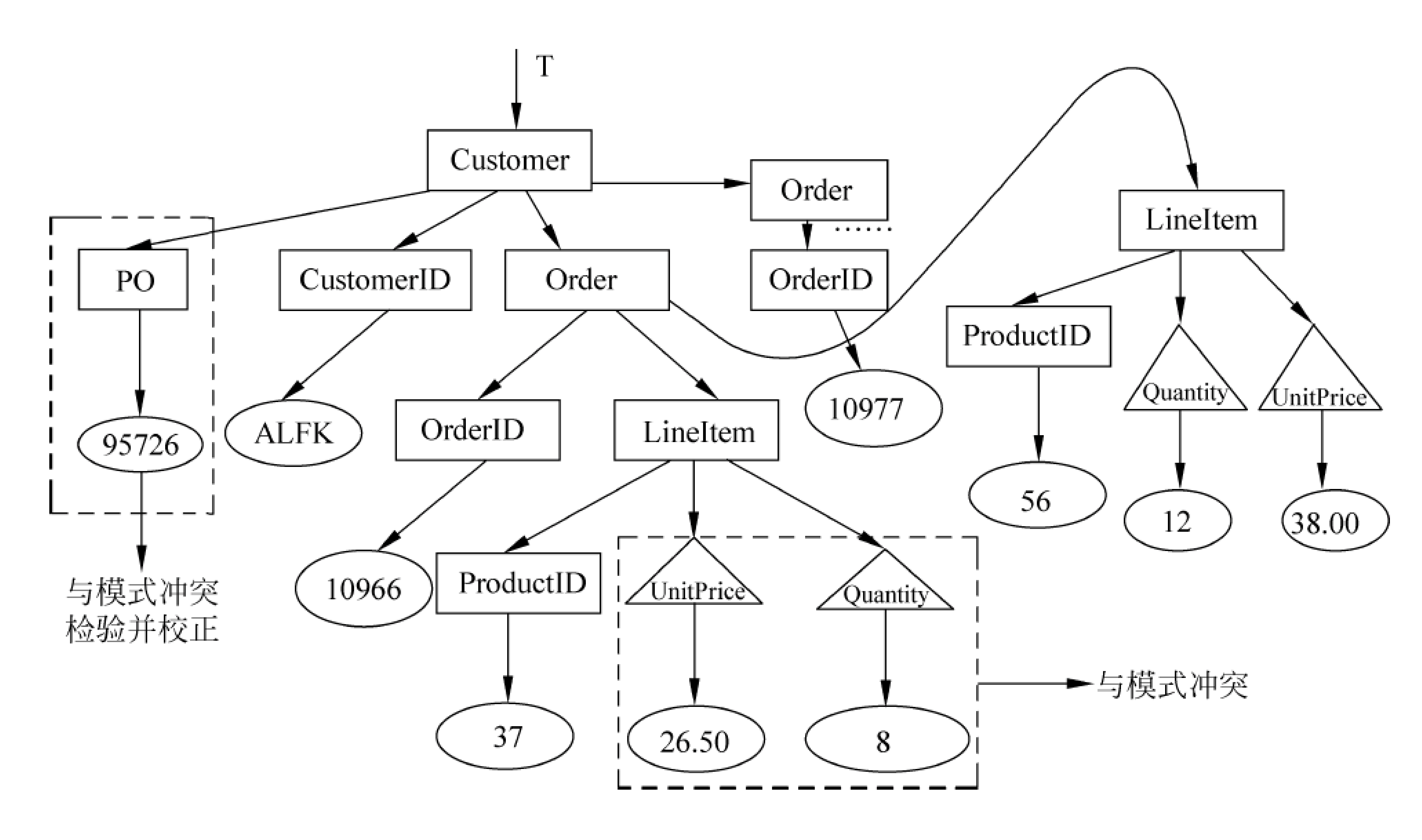


图 3-5　XML 文档树

组<size,order>,其中 order 表示节点的扩展先序遍历序号,它的取值是非连续的,主要为结点的更新操作预留空间;size 表示节点的后裔范围。<order,size>满足[24]:

(1) 对于文档树中每个分支节点 x,对于所有 x 的子节点 y,满足:$size(x) >= \sum size(y)$;

(2) 对任意给定节点 x 和 y,如果 x 是 y 的祖先节点,满足:$order(x) < order(y) <= order(x) + size(x)$。

分别就复合元素、简单元素和属性建立关系表 element_table 和 value_table,按 SchemaID、docID 字段建立 B^+-Tree 索引,叶子结点的每一项都是一个固定记录,以元素 name 为序,相同 name 以 order 为序。对应于图 3-4 的关系索引表如下,为了说明方便,这里 SchemaID、docID 都是同一个文档标识符号,故省略。

element_table(SchemaID,docID,name,order,size,level,sch_pre),对应于 ea_flag=0 的元素;

value_table(SchemaID,docID,name,order,size,level,sch_pre,value),对应于 ea_flag=1 OR 2 的元素/属性,其中 sch_pre 列对应语义模式树的模

式结构表 Schema_Structure 中相应结点的 preorder。表 3-3 和表 3-4 分别列出了图 3-5 XML 文档树定义的所有元素和属性的结点编码信息。

表 3-3 复合元素编码表 element_table

name	order	size	level	sch_pre
Customer	1	100	1	1
LineItem	15	16	3	5
LineItem	35	16	3	5
Order	10	60	2	3
Order	80	10	2	3

表 3-4 简单元素和属性编码表 value_table

name	order	size	level	sch_pre	value
CustomerID	2	5	1	2	ALFKI
OrderID	11	5	3	4	10966
OrderID	81	5	3	4	10977
ProductID	16	5	4	6	37
ProductID	36	5	4	6	56
Quantity	17	5	4	7	8
Quantity	37	5	4	7	12
UnitPrice	18	5	4	8	26.50
UnitPrice	38	5	4	8	38.00

4. 索引结构 SBXI(Schema-based XML Indexing)

基于 XML Schema 的 XML 文档数据索引和存储结构视图如图 3-6 所示，图 3-7 和图 3-8 分别是 Schema 结构索引和 XML 数据索引结构视图。

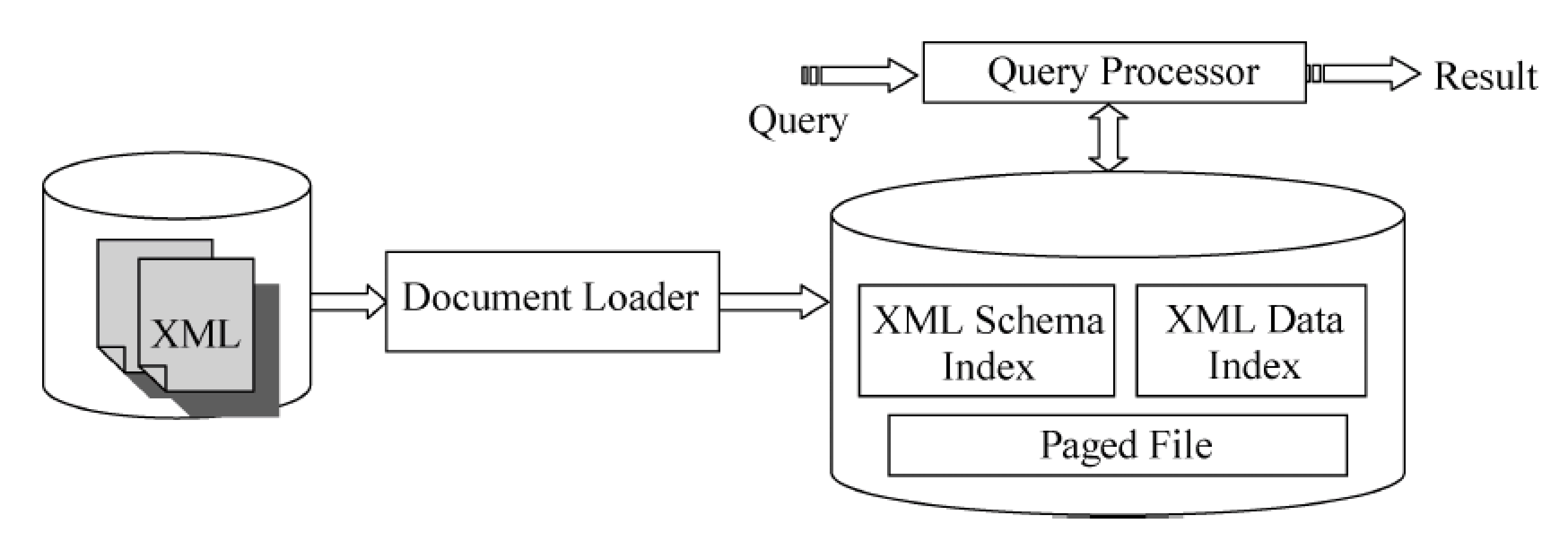


图 3-6 SBXI 结构视图

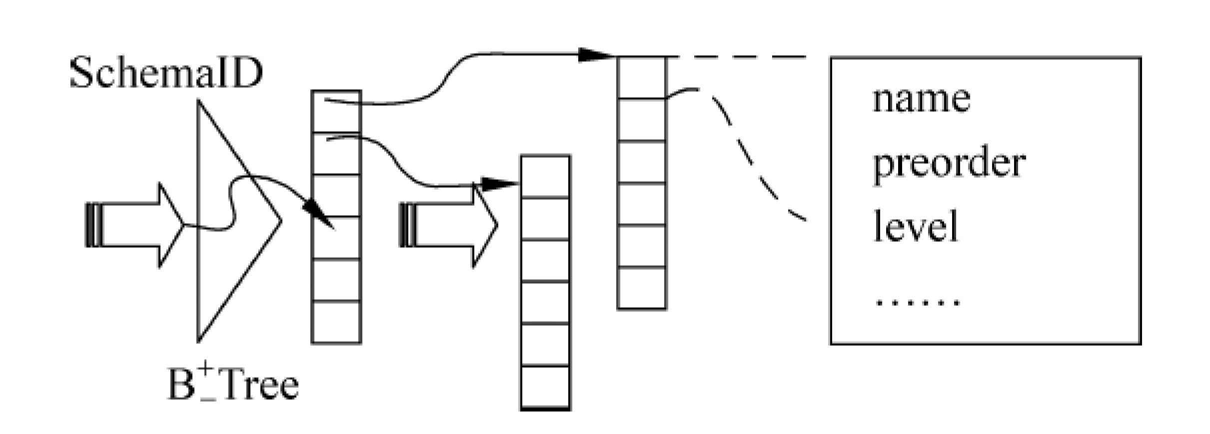


图 3-7　XML Schema 索引结构

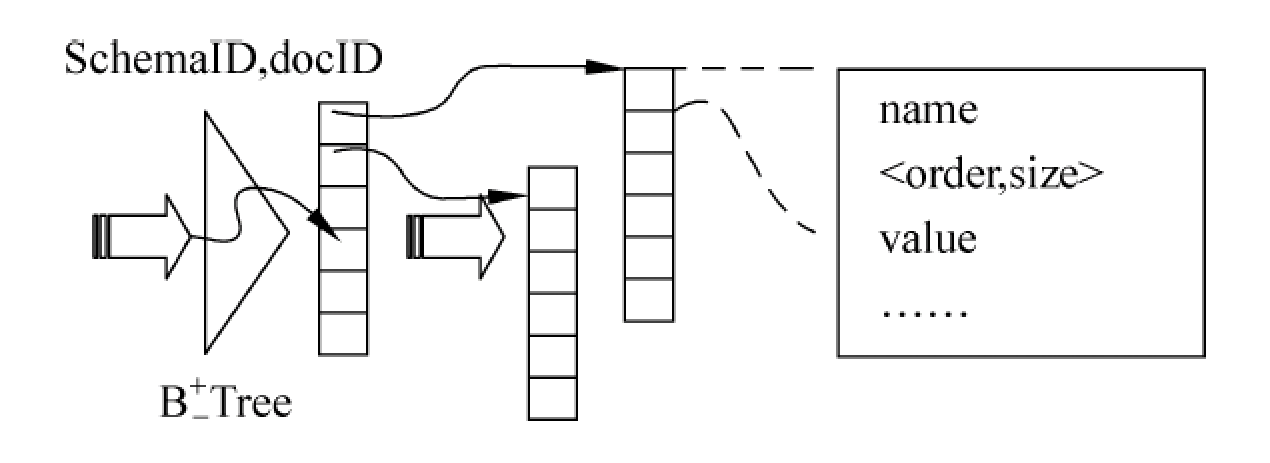


图 3-8　XML 文档索引结构

基于关系数据库存储 XML 模式信息和文档信息，可以有效利用关系数据的索引机制和物理存储，通过 sch_pre 字段将模式结构信息与文档数据信息连接起来，既节省大量的存储空间，又提高查询的路径导航效率。同时也考虑到文档数据的更新维护，采用 Li-Moon 编码，为结点的插入预留了存储空间。XML 数据文档可能很大，存储时通过＜order，size＞和 level 标识结构信息部分，从空间代价上优于 XISS[24]（XML Indexing and Storage System）结构，而且查询时避免了频繁的 JOIN 连接操作，效率上也优于 XISS 结构[17]。

如果文档数据中像 Order、LineItem 这种重复元素不存在时，完全可以省去＜order，size＞和 level 结构信息部分，只通过 sch_pre 字段就可以完全标识文档树结构。也就是说，如果一个 XML 文档中不存在同一层次的同名元素，element_table 和 value_table 可以去掉 order、size 和 level 字段信息。

3.3　基于 SBXI 存储策略的 XQuery 查询处理

XQuery 是 W3C 组织提出的 XML 查询语言标准，基于路径表达式来进行查询导航，支持含多个谓词的复杂正则表达式查询，并提供“//”查询操作

符。对于 XQuery 查询,首先进行路径有效性验证,包括路径结构信息和内容信息的检查,如路径表达式/Customer/Order[OrderID=10966]/LineItem[2],待检验的路径信息为/Customer/Order[OrderID]/LineItem。

3.3.1 查询路径有效性检验

由于 Schema 索引中记录了每个结点的父结点(parent_preorder)信息,所以对查询路径采用 down-up 算法根据元素名称进行模式匹配,如 LineItem-Order[OrderID]-Customer。如果查询路径在 Schema 中不存在,则肯定是不合法的查询路径,返回“无查询结果!”信息。为了提高查询有效性验证,可以将模式语义树中所有叶子节点的路径信息存于数据表中,便于查询路径导航。

3.3.2 XML 文档查询处理

对于有效的路径查询,SBXI 首先将路径表达式分割成多个只有一个谓词约束的路径表达式,称为简单路径表达式,分割后的每个路径表达式的查询输出结果,作为后继路径表达式运算的一个输入。例如,将上面的查询表达式分割为/Customer/Order[OrderID=10966]和/LineItem[2]两个子查询,分别对每个子查询进行文档查询处理。下面就以此例为准说明处理过程。

采用 down-up 算法处理每一个简单路径表达式,过程描述如下:

(1) 确定元素类型(type):如果有谓词查询(形如[OrderID=10966]),则查询结点一定是叶子结点,肯定元素为属性或简单元素,通过模式信息的 Schema_Structure 表的 type 列查得元素类型(如 OrderID 的 type 值为 integer)。

(2) 确定目标元素记录集合:根据(1)的元素类型,如果复合元素(complex),则在 element_table 中得到所有该元素的集合(每个元素用<order,size,level,sch_pre>表示),否则在 value_table 中根据结点名称和 value 值找到目标叶子结点集合,记为 $LF=\{lf_1, lf_2 \cdots lf_m\}$,对于不同 SchemaID 和 DocID 的结点分组处理,这里是同一 SchemaID 和 DocID,所以

结果只有一组，LF＝{＜11，5，3，4＞}。

（3）路径倒序步进（如 OrderID－＞Order－＞Customer），重复（1）和（2），找到上一层所有（如 Order、Customer）分支元素集合 BR＝{br_1，br_2，…，br_m}（此例中 BR＝{＜10，60，2，3＞，＜80，10，2，3＞}），根据以下条件在 BR 中找出所有 LF 结点的父亲结点集合 BRP。（此例中 BRP＝{＜10，60，2，3＞}，因为 11＞10）

条件：$order(br_i)< order(lf_j)<= size(br_i)+ order(br_i)$ and $level(lf_i)= level(br_j)+1$

（4）重复过程（3），直到处理完一个简单路径表达式。（查询结果为 Customer$_1$-Order$_{10}$-OrderID$_{11}$，角标表示 order 序列值）

（5）同样处理其他子查询，如/LineItem[2]表示上下文结点的第 2 个子结点 LineItem，找出所有 LineItem 结点记录，按 order 值升序排序，则第 2 个 LineItem$_{35}$即为查询结果。

（6）子查询的连接：邻接两个子查询的连接运算采用 up-down 算法，使用上面的条件进行结构连接，产生最后结果。

SBXI 存储和索引结构支持“//”操作符，例如对于形如/Order//LineItem 的查询，只要找出所有 LineItem 的目标元素集合 LF＝{＜15，16，3，5＞，＜35，16，3，5＞}和 Order 元素集合 BR＝{＜10，60，2，3＞}，再分别根据＜order，size，level＞的条件值进行结构连接，即可完成查询。

从查询过程可以看出，由于文档信息编码中含有结构信息，通过 sch_pre 将结点的 order 和 preorder 连接起来所以大大减少了结构连接的次数，同时，只通过遍历 Schema 索引表就能检验查询路径的有效性，提高了查询效率。

3.4　基于关系存储的 XML 文档更新

良好的数据存储格式、可扩展性、高度结构化以及便于网络传输等特点，决定了 XML 卓越的性能表现。通过网络实现企业间的信息交换是企业进行电子商务的一种重要手段，随着电子商务的广泛应用，XML 已经成为

事实上的信息交换和信息整合的标准,能有效并且高效的存取 XML 的信息相应的也变得越来越重要。要做到这一点,必须要有一个能够准确的获得所需信息、更新 XML 数据源中数据的可表达的查询语言。W3C 于 2001 年提出结合了 XML 主流查询语言许多优点的 XQuery 语言[26,27],但是 XQuery 标准没有提供在一个持久的 XML 文档上进行更新操作的机制,也没有 XML 商业系统支持通过 XQuery 对 XML 文档进行更新。但是对于许多电子商务信息系统,高效的更新操作不可或缺。本节的主要工作是在 XQuery 查询数据模型的基础上,设计 Update 语句的语法和语义,在此基础上对 XML 元数据的存储模型及更新机制提出实现逻辑。

虽然人们提出了很多存储方法和查询优化措施,但作为完整的 XML 数据库管理系统缺少更新功能是不现实的,而目前在这方面的研究很少。一些基于对象管理器的数据库系统如 Lorel,提供了对 Update 的支持,但都是基于编程语言的操作序列,涉及多个对象的插入和删除,十分复杂。文献[28]提出的 XML-RL(XML Rule-based Query Language)语言对更新部分的定义包括两部分:查询和修改,支持多变量绑定和有序多文档修改,但没有表述元数据的存储模型及实现机制。文献[29]不仅给出了修改 XML 文档的语法定义,而且在关系存储基础上分析了各种修改策略的性能。但是其存储模型带来大量的外联操作,对于频繁更新的 XML 数据管理代价太高。文献[30、31]重点指出了 XML Update 语言除了能够查询和修改文档内容的基本要求外,语言本身必须简洁、直接面向用户,而且必须支持 XML 命名空间(XML Name Spaces)和 Xpath 技术,本节的研究重点放在语言定义的简洁性和实现机制的开放性、实用性。

3.4.1 基于扩展 XQuery 数据模型的文档更新操作

本节定义的基于扩展 XQuery 的 XML 更新语言 XML-XUL(XQuery-based XML Update Language)不仅考虑对无序文件更新支持,同时考虑对有序文件进行更新的需要,并支持 IDREFS 操作,尽量使语法规则与 XQquery 的 FLWR 表达式相容,同时与 SQL(Standard Query Language)接

近，使其成为 XML 的 SQL。

1. 扩展 XQuery 数据模型

XQuery 数据模型将 XML 文档视为带有标签的结点树，如图 3-10 就是图 3-9 中 XML 文档对应的文档树。

```
<Customer>
  <CustomerID>ALFKI</CustomerID>
    <Order>
      <OrderID>10966</OrderID>
      <LineItem>
        <ProductID>37</ProductID>
          <UnitPrice>26.50</UnitPrice>
    </LineItem>
    <LineItem>
      <ProductID>56</ProductID>
      <UnitPrice>38.00</UnitPrice>
    </LineItem>
    </Order>
</Customer>
```

图 3-9　XML 文档片段

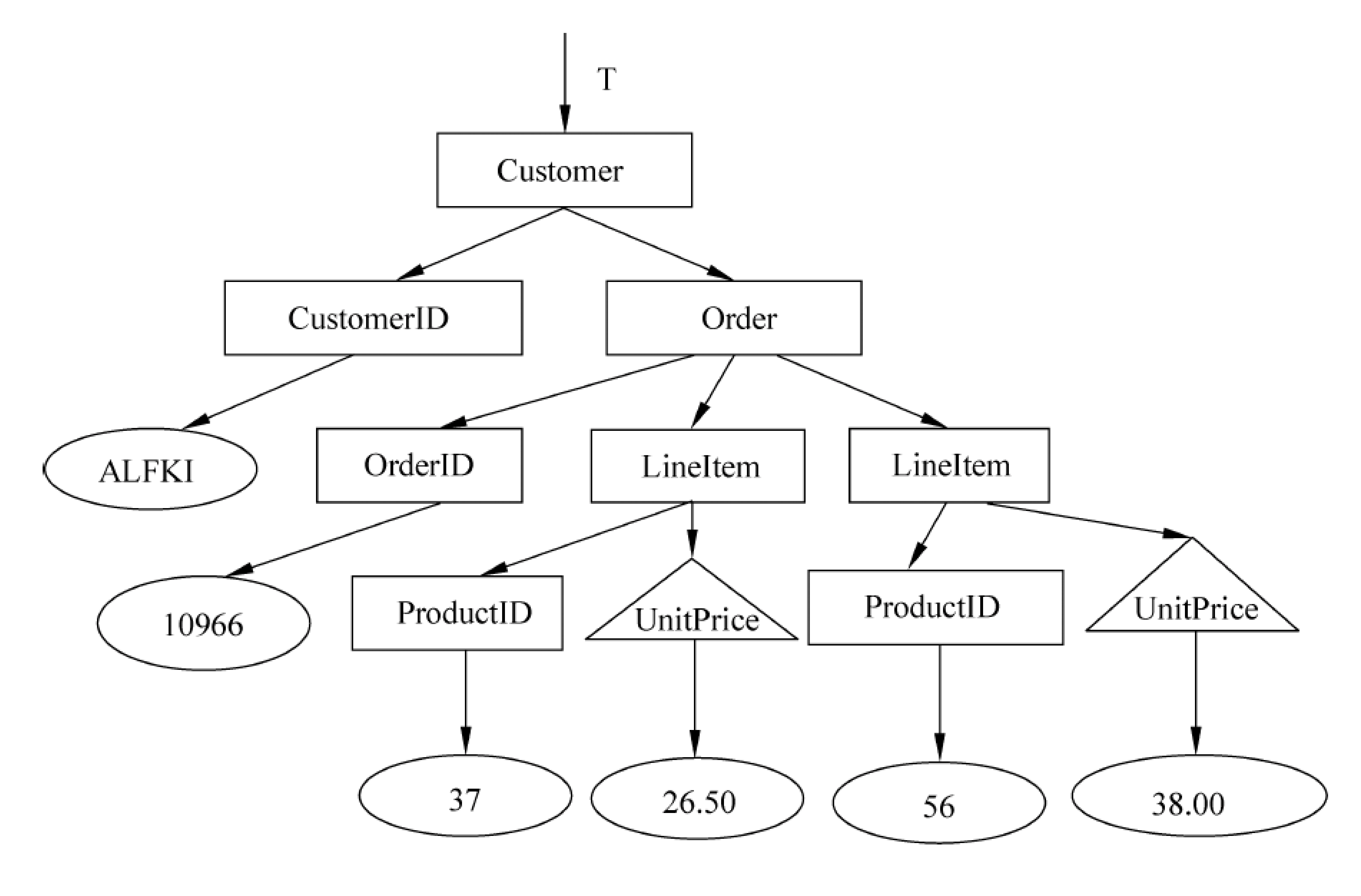


图 3-10　XML 文档树

文档更新的粒度可以是文档内容中的组成部分，主要包括以下 6 项内容：

(1) 属性(Attribute)：属性表示为(属性名，属性值)的有序对，为了与元素区分，在属性名前添加符号@，如(@UnitPrice，26.50)。

(2) 元素(Element)：元素表示为由元素名和一系列子元素或属性构成的序列，通过符号"－＞"表示元素与子元素之间的关系，如(LineItem－＞ProductID，@ UnitPrice)。

(3) 文本(CDATA)：指 CDATA 类型的大文本数据。

(4) 处理指令(Processing-Instruction)：表示为(处理指令名，处理指令信息)，如＜?cocoon-process type＝"xsp"?＞指令表示为(cocoon-process，"xsp")。

(5) 标识符引用(IDREF、IDREFS)：IDREF 类型属性的值引用 ID 类型属性值，隐含父子关系，表示为(@IDREFS 属性名称－＞@ID 属性值 1 @ID 属性值 2 ……)。

(6) 注释(Comment)：类似于文本的字符串值。

2. XUL 操作

XUL 提供的更新子操作 UpdateOp 包括：

(1) Delete(node)：删除目标对象 node，node 可以是上述 6 项内容之一，如果 node 是一个复合元素，自动先行将子元素或属性删除；如果是一个被引用的 ID，也要将引用删除；如果是 IDREF，只是将 IDREFS 中的该引用移走。

(2) Rename(node，name)：将目标对象 node 更名为 name，Rename 不能操作 CDATA 和 IDREFS 中的一个 IDREF 值，但可以对整个 IDREFS 重新命名。

(3) InsertBefore(ref，content)：仅应用于有序文档。如果 ref 是目标对象的一个元素或者 CDATA(不能为属性，因为属性无序)，那么 content 也必须是一个元素或 CDATA，将 content 直接插在 ref 的前面；如果 ref 是

IDREFS,则 content 必须是一个 ID,将 ID 插在{ref}. position(i)之前。position(i)可以是系统或用户定义的函数。

(4) InsertAfter(ref,content):与 InsertBefore(ref,content)定义类似,不同是将 content 直接插在 ref 的后面。

(5) Append(content):仅用于无序文档。在目标对象尾部追加 content,content 可以是上述 6 种内容之一,但插入属性时 content 是不能与已有属性同名。

(6) Replace(node,content):对于有序文档,等价于先 InsertBefore(node,content)操作然后执行 Delete(node),对于无序文档,等价于先执行 Append(content)然后执行 Delete(content)操作。

3.4.2　XUL 操作语义和实例

1. XUL 的 FLWR(FOR-LET-WHERE-RETURN)语法

将以上 6 个操作映射为 XQuery 语言的 FLWR 语法,用 EBNF 描述为:

```
FOR $binding1 IN XPath-expr, …
LET $binding  := XPath-expr, …
WHERE predicate1, …
UPDATE $binding1 {subOp {, subOp} * }
```

这里的 *subOp* 指 6 种原语子操作,描述如下:

```
DELETE $node |
RENAME $node TO name |
INSERTcontent BEFORE|AFTER $node |
APPEND content|
REPLACE $child WITH $content|
FOR $binding' IN XPath-subexpr, …
WHEREpredicate1, . . . updateOp
```

UPDATE 定义允许嵌套,以便在复合 XML 结构内实现多层次更新操作。

2. XUL 操作细则和实例

示例 1：删除操作。假定图 3-11 中文档片段来自文件 ware. xml，文档根元素为 Customer，下面语句完成删除单价小于 30 的 LineItem 元素。

```
FOR $ item INdocument("ware.xml")//LineItem
LET $ price := $ item/UnitPrice
WHERE $ price<30.00
UPDATE $ item {
    DELETE $ item->ProductID
    DELETE $ item->@UnitPrice
    DELETE $ item }
```

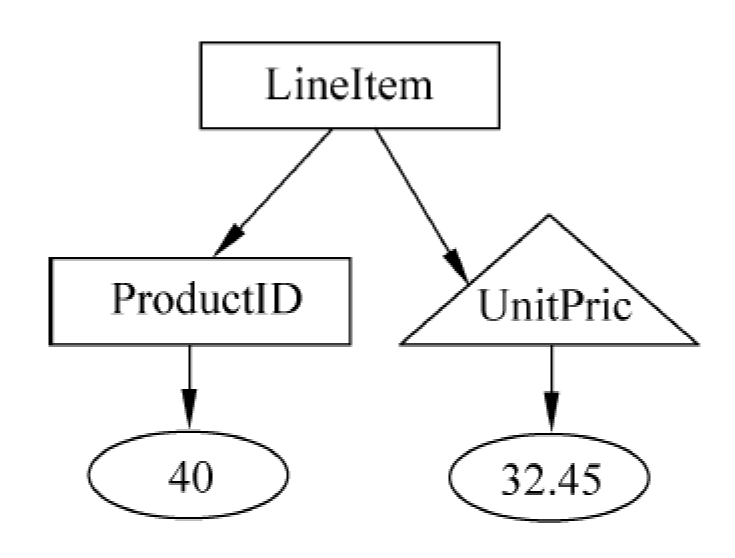


图 3-11 插入 LineItem 元素

示例 2：插入操作。下面语句完成给 ID 为“ALFKI”的 Customer 元素插入一个属性(@PO，“9572658”)。

```
FOR $ c INdocument("ware.xml")/Customer[CustomerID="ALFKI"]
UPDATE $ c{
    APPEND new_attribute(@PO, "9572658")}
```

示例 3：插入一个如图 3-11 所示的 LineItem 元素，保证 ID 有序。

```
FOR $ item INdocument("ware.xml")//LineItem
LET $ p :=$ item/ProductID
WHERE $ p>=40.00
UPDATE $ item{
    new_element(ProductID, 40)
    new_attribute(@Unitprice, 32.45)
    INSERT (LineItem->ProductID, @Unitprice ) BEFORE { $ item }. first()}
```

也可以使用 INSERT AFTER 操作，筛选所有 ProductID< 40 的

LineItem 元素，在最后一个元素的后面插入，即可完成有序插入。

示例 4：重命名操作。下面语句完成将所有 OderID 元素的名字改为 OderNo。

```
FOR $o INdocument("ware.xml")/Customer/Order/OrderID
UPDATE $o{
    RENAME $o TO OderNo}
```

示例 5：嵌套应用。有时需要一次对多个层次的元素进行修改，就需要 XUL 的嵌套使用了。如：

```
FOR $c INdocument("ware.xml")/Customer
    $order IN $c/Order
UPDATE $c {
    new_ref(@PONo->t001)
    APPEND new_attribute(@PO,@PONo)
    FOR $item IN $order/LineItem[2]
    UPDATE  $item{
        REPLACE 40.00 WITH $item->@UnitPrice}}
```

此语句在 Customer 结点插入一个引用属性@PO，指向一个 ID 属性@PONo，值为 t001，同时在将第 2 个 LineItem 元素的 UnitPrice 属性值变更为 40.00。

3.4.3　基于触发器机制的更新实现

为了保证数据的完整性和一致性，更新操作必须是“安全”“有效”的，这就需要更新粒度层的锁定原则和数据类型及数据路径结构的一致性检查。

DTD[W3C 1998]和 XML Schema[W3C 2001b]定义了文档结构，以此作为“有效”检查的模式规范。本小节从两个方面实现在数据更新过程中的先验过程[32]，逻辑架构图如图 3-12 所示。

1. XQuery 处理器的语法分析

Xquery 处理器主要完成 XUL 语句的语法分析，同时根据 XML Schema 模式信息进行数据类型和路径结构匹配的有效性验证，如元素出现的次数、

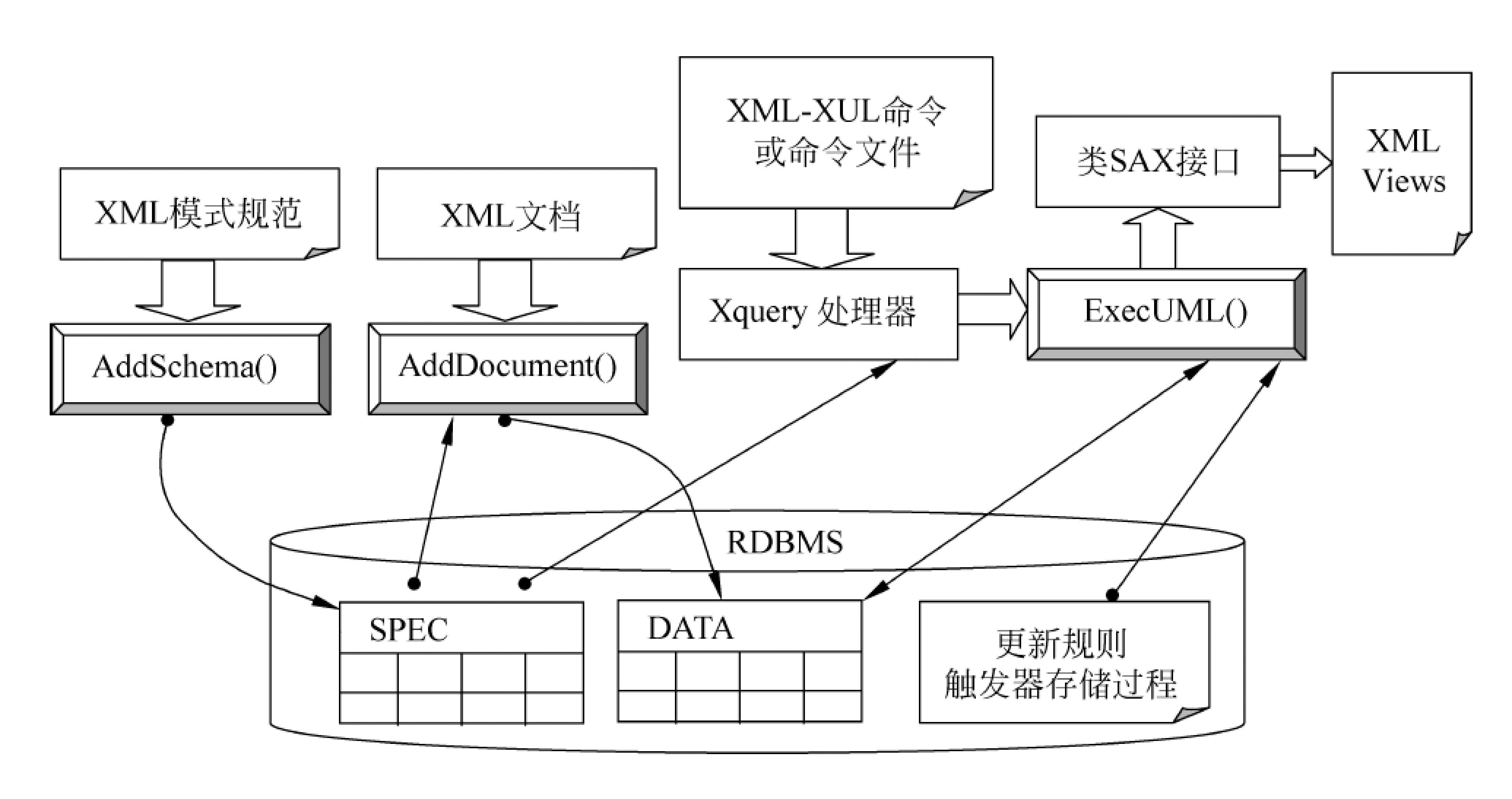


图 3-12 XML 文档更新实现的逻辑架构

谓词操作符的类型匹配、检索路径的合法性等。

2. 更新执行时自动触发更新规则

根据 XML 文档映射到关系数据库中的规则确定更新锁定粒度，如果映射为多表，则锁定粒度为表；如果映射到单表(可以根据元素值的类型对表进行横向分割)，则锁定粒度为记录。3.2 节讲述了如何对文档结点采用 Li-Moon 编码进行表存储而且有利于文档更新的实现策略。

针对存储映射方法制定一套 ECA(Event Condition Action 事件—条件—动作)规则，类似于关联规则的更新规则，定义为相应的触发器，在更新语句执行时自动触发。例如试图删除具有元素或属性的复合元素时，应该先行删除其子元素；再如试图删除标识符引用时(IDREF)，只能从 IDREFS 列表中将该项移走，特别是更新嵌套时，有些操作是严格有序的。

3. 更新操作的实现

更新操作的实现是在 University Pennsylvania 的 XML 查询系统 Kweelt[33]的基础上，建立扩展 XQuery 实现对 XML 文档更新的系统。工作流程是：输入 XQuery 更新操作语句，即 XML-XUL 命令，通过 XQuery 处

理器获取一个文本格式的先验 XQuery 更新语句，通过 ExecUML()处理部件执行获取的新语句。上述步骤使用了两个部件来解决传统更新操作中的问题。一个部件是 XQuery 处理器，它接受一个 XQuery 更新语句，即一个 FLWR 表达式，输出一个添加了有效性判断的 XQuery 更新语句；第二个部件是 XQuery 更新执行引擎 ExecUML()，由 Kweelt 系统修改扩展而来。XQuery 更新执行引擎接受三个输入，一个 XML 和 XML Schema 文档、一个符合 XQuery 更新语法的 XML 更新操作语句和基于触发器的更新规则。其中更新语法是扩展得到的。如果用户不需要 XML 数据加入限制条件，可以不输入 XML Schema 文档。同时 XML 更新操作语句也可以不经过先验 XQuery 生成器得到，但这样无法保证有效性。两个部件功能合在一起才能形成功能完全的 XML 更新系统。

对 Kweelt 系统中的 Java 类进行修改，使其可以识别更新语法中的关键字，同时加入执行这些关键字操作的类和方法。

第一步：使 Kweelt 系统能理解扩展的 XQuery 更新语法。通过 JavaCC 文件 QueryParser. JJ(Java Compiler Compiler 文件，纯 Java 的词法分析生成器)存放 Kweelt 系统中用到的 XQuery 语法，JavaCC 通过 QueryParser. JJ 文件生成新的 Java 类，用这些 Java 类来实现对输入 XQuery 查询的解析，生成语法树。修改 QueryParser. JJ 文件使其识别 XQuery 更新语法，加入文件中的更新语法用函数的方式定义，与可能输入的各 Query 更新操作相对应。

第二步：扩展 Kweelt 系统函数库，定义所有更新操作所需的函数，函数中包含类似 EBNF 的语法和使用特定标识对其他函数及 Java 代码的调用。QueryParser. JJ 文件中一个包含 FLWR 表达式的函数如下：

```
QueryExpression flwr_expression()
{
    QueryExpression e;
    Vector sort= null;
    QueryExpression where=null;
    Binding b;
    Vector bindings=new Vector();
```

```
}    //构造 FLWR 表达式类
{
    <FOR> b= for_binding(true) {bindings.addElement(b); }
    (<COMMA > b=let_ binding( false)
        {bindings.addElernent(b); }     //在 LET 语句中对绑定变量 b 的处理
    [<where>where=where_clause()]           //对 where 子句的处理
        <RETURN > e=query_expression()  //对 return 子句的处理
      [sort=sort_clause_list()]             //对 sort 命令的处理
    return new FLWRExpression(bindings, where, e, sort); }
      //返回经过处理解析的 FLWR 表达式
}
```

参考文献

[1] Quass D, Widom J, Goldman R, Haas K, Luo Q, McHugh J, Nestorov S, Rajaraman A, Rivero H, Abiteboul S, Ullman J, Wiener J. LORE: A lightwight object repository for semistructured data. In: Jagadish HV, Mumick IS, eds. Proc. of the 1996 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 1996. 549～554

[2] Floreseu D. KossmanD. Storing and Querying XML Data using an RDBMS. IEEE Data Engineering Bulletin, September 1999, 22(3): 27～34

[3] Mchugh J, Abiteboul S, Goldman R, et al. Lore: A Database Management System for Semistructured Data. ACM SIGMOD Record, 1997, 26(3): 54-66

[4] Zhou AY, Lu HJ, Zheng SH, Liang YQ, Zhang L, Ji W, Tian ZP. VXMLR: A visual XML-relational database system. In: Apers P, Atzeni P, Ceri S, Paraboschi S, Ramaohanarao K, Snodgrass R, eds. Proc. of the 27th Int'l Conf. on Very Large Data Bases. San Fransisco: Morgan Kaufmann Publishers, 2001. 719～723

[5] Wang Q, Zhou JM, Wu HW, Xiao JC, Zhou AY. Mapping XML documents to relations in the presence of functional dependencies. Journal of Software, 2003,14 (7): 1275～1281. http://www.jos.org.cn/1000-9825/14/1275.htm

[6] Tatarinov I, Viglas S, Beyer K, Shanmugasundaram J, Shekita E, Zhang C. Storing and querying ordered XML using a relational database system. In: Franklin M, Moon B, Ailamaki A, eds. Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2002. 204～215

[7] Zhang C, Naughton J, DeWitt DJ, Luo Q, Lohman G. On supporting containment queries in relational database management systems. In: Aref W, ed. Proc. of the 2001 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press,

2001. 426～437

[8] Arenas M, Libkin L. A normal form for XML documents. ACM Trans. on Database Systems, 2004,29(1): 195～232

[9] Christophides V, Cluet S, Moerkotte G, Siméon J. On wrapping query languages and efficient XML integration. In: Chen W, Naughton J, Bernstein P, eds. Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2000. 141～152

[10] Ludascher B, Papakonstantinou Y, Velikhov P. Navigation-Driven evaluation of virtual mediated views. In: Zaniolo C, Lockemann P, Scholl M, Grust T, eds. Advances in Database Technology-EDBT 2000, 7th Int'l Conf. on Extending Database Technology. Berlin, Heidelberg: Springer-Verlag, 2000. 150～165

[11] Buneman P, Fernandez M, Suciu D. UnQL: A query language and algebra for semistructured data based on structural recursion. The VLDB Journal, 2000,9 (1): 76～110

[12] Beeri C, Tazban Y. SAL: An algebra for semistructured data and XML. In: Cluet S, Milo T, eds. Proc. of the 2nd ACM SIGMOD Workshop on the Web and Databases. INRIA, 1999. 37 ～ 42. http://www-rocq. inria. fr/～ cluet/webdb99. html

[13] Jagadish H V, Lakshmanan L V S, Srivastava D, et al. TAX: A Tree Algebra for XML. In: Clark J et al Eds. Proceeding of the International Workshop on Database Programming Languages (Lecture Notes in Computer Science, Vol. 2397). Rome, Italy, September 8-10, 2001. Heidelberg: Springer - Verlag, 2002. 149～164

[14] Fernandez M, Simeon J, Wadler P. An Algebra for XML Query. In: Kappoe S et al Eds. Proceeding of the 20th FSTTCS International on Foundation of Software Technology and Theoretical Computer Science (Lecture Notes in Computer Science, Vol. 1974). New Delhi, India. December 13-15, 2000. Springer-Verlap, 2000. 11～45

[15] Meng XF, Luo DF, Jiang Y, Wang Y. OrientXA: An effective XQuery algebra. Journal of Software, 2004, 15 (11): 1648 ～ 1660 (in Chinese with English abstract). http://www. jos. org. cn/1000-9825/15/1648. htm

[16] Xuebin Chen, J. Duan, Guolin, Hongcan Yan. Schema-based constrained XML data indexing and storage technique, NISS 2009,2009. 6: 973-978

[17] Yan Hongcan, Li Minqiang. The Definition and Implementation of XML Document Update Language Based on Xquery. DCABES 2008: 210-219

[18] Nagy M,Walsh N,et al. XQuery 1. 0 and XPath 2. 0 Data Model. Editors [EB/OL]. W3C. 4, Apr 2005. http://www. w3. org/TR/xpath-datamodel/6 World Wide Consortium. Extensible Markup Language (XML)1. 0

[19] Tian Feng，De WittD J，et al. The Design and Performance Evaluation of Alternative XML Storage Strategies. SIGMOD record，March 2002，31(1)：5～10
[20] Michel Stonebraker. 对象-关系数据库管理系统. 杨冬青等译校. 北京：北京大学出版社，1997
[21] Akmal B. Chaudhri，Awais Rashid，Roberto Zicari. XML 数据管理纯 XML 和支持 XML 的数据库系统. 邢春晓，张志强等译. 北京：清华大学出版社，2006
[22] Brett Mclaughlin. JAVA 与 XML. 孙兆林等译. 北京：中国电力出版社. 2001
[23] Goldman，R.，Widom，J.：DataGuides：enabling query formulation and optimization in semistructured databases. In Proc. of the Int'l Conf. on Very Large Databases，1997. 436～445
[24] Quanzhong Li，Bongki Moon，Indexing and Querying XML Data for Regular Path Expression[C]. Proceedings of the 27th International Conference on Very Large Database，Roma，Italy，September 11-14，2001. San Francisco：Morgan Kaufmann Publishers，2001：361-370
[25] Fankhauser P，*et al*. XQuery 1.0 and XPath 2.0 formal semantics [EB/OL]. 2005. http://www.w3.org/TR/query-semantics/
[26] Chamberlin D，*et al*. XQuery 1.0：An XML query language[EB/OL]. 2005. http://www.w3.org/TR/xquery
[27] S. Abiteboul，D. Quass，J. McHugh，J. Widom，and J. L. Winer. The Lorel query language for semistructured data. In Proceedings of International Journal on Digital Libraries，volume 1(1)，April 1997. 68～70
[28] Mengchi Liu，Li Lu and Guoren Wang，A Declarative XML-RL Update Language. ER 2003，LNCS 2813，2003. Springer-Verlag Berlin Heidelberg 2003. 506～520
[29] Tatarinov I.，Ives Z. G.，Halevy A. Y.，Weld，D. S.：Updating XML. In Proceedings of 2001 SIGMOD Conference，Santa Barbara，CA，USA (2001). 413～424
[30] Andreas Laux. XML Update language[EB/OL]，Working Draft-2000-09-09，http://www.infozone-group.org/lexusDocs/html/wd-lexus.html
[31] Lars Martin. XML Update Language Requirements[EB/OL]，Working Draft-2000-11-24，http://xmldb-org.sourceforge.net/xupdate/xupdate-req.html
[32] 阎红灿，王淑芬等. 基于 XQuery 数据模型的 XUL 语言的定义和实现[J]. 东北师大学报. 2008，40(4)：52-57
[33] Sahugust A. Kweelt：the Making-of Mistakes Made and Lessons Learned [EB/OL]. 2000.11，http://db.cis.upenn.edu/Publications/2000

第 4 章　基于频繁模式挖掘的 XML 网页分类技术

要实现 Web 的分类检索，首先解决网页分类问题。XML 文档是文本内容信息与结构信息的综合体，XML 文档分析区别于传统的文本分析的关键在于结构信息的获取与利用，针对一般文件的分类方法不但体现不出 XML 文件的优点，更可能使得分类效果相对于普通文本效果更差，因此有必要研究一种专门针对 XML 文档的分类方法。

近年来，国内外研究者对 XML 文档等半结构化数据的分析处理给予了越来越多的关注，但文本挖掘方面的研究成果相对较少。Yi 等人提出了一种用于半结构化文档分类的扩展向量模型，采用嵌套定义的向量描述文档元素，通过概率统计方法进行分类[1]，Denoyer 等人提出了利用贝叶斯网络模型进行半结构化文档分类的方法[2]，Zhang 等人采用编辑距离计算 XML 文档相似性[3]，Flesca 等人将结构信息看作时序关系，通过时序分析计算文档结构相似性[4]，Zaki 等人通过挖掘频繁子树[5]提出了代价敏感的结构规则抽取方法[6]，对半结构化数据分类取得较好的效果。但是这些方法仅研究文档的结构关系，没有考虑文本内容，或者针对特定的挖掘技术，虽然想法新颖，但推广性较差。

本章在 XML 文档结构相似度量[7-8]基础上，结合 XRules[5]规则、无序频繁子树挖掘算法[9]和 N 层向量空间模型[10-11]提出一种综合利用 XML 分层结构信息和关键词内容信息[12-14]进行文档分析的数据模型—频繁结构层次向量模型（Frequent Structure Hierarchy Vector Model，FSHVM），通过构造决策表，利用粗糙集理论的属性约简达到特征值降维目的[15-19]，实现了基于规则的分类方法。实验表明，此方法不仅具有更高的准确性，而且计算代

价更小。

4.1 频繁模式挖掘算法 TreeMiner$^+$

在频繁结构向量模型中,将数据库文档集合中所有频繁子树(模式)视为文档结构的特征空间,相当于传统的向量空间模型中的关键词特征。在传统的向量空间模型中,如何抽取特征关键词是一个关键的技术问题,而在频繁结构向量模型中,如何挖掘频繁模式也是一个同等重要的技术问题。

4.1.1 频繁模式挖掘算法 TreeMiner

挖掘频繁子树的算法[20]是基于最右扩展递增模式的,基本思路是:首先获得 1-Subtrees,即含有一个结点的子树,通过计算每个标签的支持度,选出候选频繁结点。然后通过共享前缀的类结点右扩展生成 2-Subtrees,通过类结点的范围列表连接计算子树支持度从而选择候选类,为下一层的类扩展做数据输入。依次递归由 k-频繁子树生成(k+1)-频繁子树,直至产生所有频繁子树。

1. 频繁子树扩展的相关技术

由 k-频繁子树生成($k+1$)-频繁子树时,首先需要确定扩展类,即候选扩展结点集合。假定 P 指向 k-频繁子树前 $k-1$ 个结点的前缀编码(prefix),以$[P]_k$ 表示其类结点集合,其中每个元素形如(x,i)形式,x 表示扩展结点的标签,i 表示扩展结点的扩展位置。

性质 4-1 类扩展(Class Extension):(x,i)和(y,j)为$[P]_k$ 类集合中的任意两个结点元素,Px 表示(x,i)的扩展类,两个结点的连接运算定义为 $x \otimes y$,运算具有以下性质:

(1) 当 $i=j$ 时,如果前缀 P 不为空,在[Px]中添加结点(y,j)和($y,j+1$),如果为空,只添加($y,j+1$);

(2) 当 $i>j$ 时,在[Px]中添加结点(y,j);

（3）当 $i<j$ 时，不作任何操作。

性质 4-2 范围列表连接（Scope-List Join）：[Px]类集合中每个元素都有自己的范围列表，以£(x)表示，集合中每个元素以三元组编码（见定义 4-3），范围列表的连接运算定义为£(x) ∩⊗£(y)，具有以下性质：

（1）如果 $ty=tx=tid$，则两个结点 x、y 是在同一文档树 T 中。

（2）如果 high(x)>=high(y)，则在树 T 中 x 不是 y 的祖先结点。

（3）如果 $lx<=ly$ 且 $ux>=uy$，即 scope(y)⊂ scope(x)，说明在树 T 中 y 是 x 的后代结点，此时进行内部连接（in-scope link）。在[Px]类扩展结点的范围列表中添加结点（ty，{match labels∪lx}，scope(y)）（此时范围列表与频繁结点的范围列表有些区别，没有了层次信息，添加了匹配标签）。

（4）如果 $ux<=ly$，即 scope(y)<scope(x)，说明在树 T 中 y 是 x 的兄弟结点，此时进行外部连接（out-scope link）。在[Px]类扩展结点的范围列表中添加结点（ty，{match labels∪lx}，scope(y)）。

2. TreeMiner 频繁子树挖掘算法

```
TreeMiner (D, minsup):
F1 = { frequent 1-subtrees};
F2 ={ classes [P]_1 of frequent 2-subtrees};
for all  [P]_1 ∈ E do Enumerate-Frequent-Subtrees([P]_1);
Enumerate-Frequent-Subtrees([P]):
for eachelement (x; i) ∈ [P] do
[Px] =ϕ;
for eachelement (y; j) ∈ [P] do
R ={(x; i) ⊗ (y; j)};
L(R) = {L(x) ∩⊗ L(y)};
if for any R ∈ R, R is frequent then
[Px] = [Px] ∪{R };
Enumerate-Frequent-Subtrees([Px]);
```

上述为 TreeMiner 算法高层结构描述，主要包括频繁项 F1（1-subtrees）的计算和 2 级频繁子树（2-subtrees）F2 的计算，然后通过深度优先枚举每个类$[P]_1 \in F2$ 的其他频繁子树，是一个递归调用的过程。当然，假定初始数据库是一个已经水平编码的形式（详细内容请参阅文献[20]）。

4.1.2 TreeMiner 算法的改进

TreeMiner 算法的时间和空间复杂性与数据库 D 的大小关系很大，计算 F1 时，使用一维数组；计算 F2 时，使用矩阵表示 $F1\times F2$，时间复杂性为每个文档树 $O(n^2)$，其中 $n=|T|$。计算 Fk 时，如果类[P]集合有 n 个元素，则时间复杂性将是 $O(ln^2)$，其中 l 是范围列表的长度。算法的空间复杂性主要与初始范围列表 Scope-List 和当前类[P]及候选类[Px]的存储有关。为了使之适合大文档集合的频繁子树挖掘，提高挖掘效率，我们对其进行改进，以降低时间复杂性。

对 TreeMiner 算法在两个方面进行了改进：①频繁结点的数据结构增加了层次信息编码，有利于范围列表连接时快速判断其祖先-后代关系，只有£(y)中结点是£(x)结点的后代结点时才有可能进行内联和外联的测试，这样排除了大量冗余运算，提高了算法性能。如结点 1 与结点 4 构成 2-Subtree(1,4,−1)，但是结点 4 与结点 1 不是嵌入子树，不需要连接操作；②本着“先预算支持度，后进行类扩展”的原则，将类扩展的结点连接和频繁子树支持度的范围列表连接运算顺序重新做了调整，这样对不存在的子树或者不频繁的子树不再进行类扩展运算，大大提高了挖掘效率。

改进的频繁子树挖掘算法：

```
TreeMiner+ (D, minsup)
Input: triple code set D of document trees, minimum support of frequent subtree
Output: prefix code of all frequent subtrees S andδT(S)
compute support of each label, choose frequent labeled nodes F1;
built candidate class extension set P0={(F1, −1)};
prefix[P0]={};code[P0,k]= (F1, −1);
for each element of [P0] do Enumerate-subtrees([P0]);
Enumerate-subtrees([P])
{ For each element (x,i) ∈[P] do
    { [Px]=ϕ;
      For each element (y,j) ∈P do
      if high(x)>high(y) then                //测试 y 是 x 的后代结点
          {£(S) =£(x) ∩⊗£(y);                //x 结点与 y 结点的连接运算
          if S is frequent then              //S 是否频繁
```

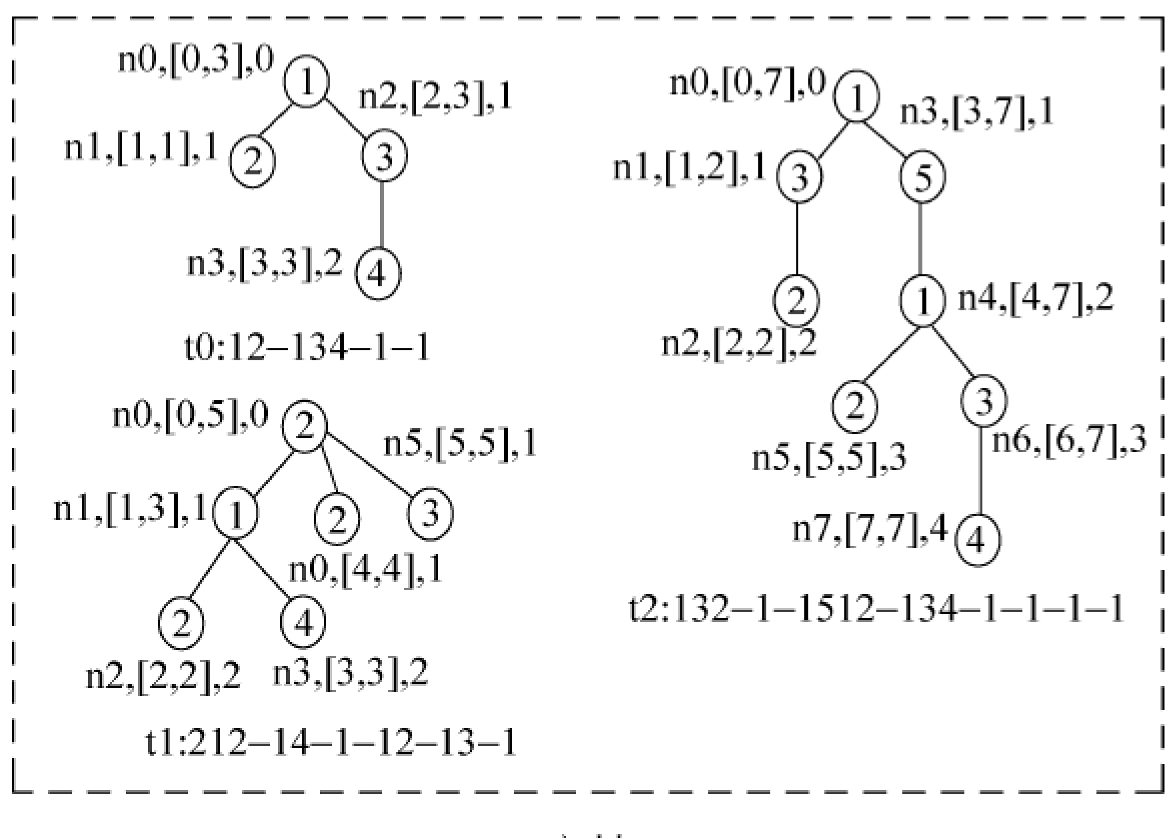
```
12:             {R=(x,i)⊗(y,j)
13:               [Px]= [Px]∪{R};            //扩展 Px 类
14:                 k+=1;
15:                 build prefix[Px] and code[Px,k];
16:             }
17:         }
18:    Enumerate-subtrees([Px]);             //递归调用
19:    }
20: }
```

4.1.3　TreeMiner^{+}算法挖掘处理实例

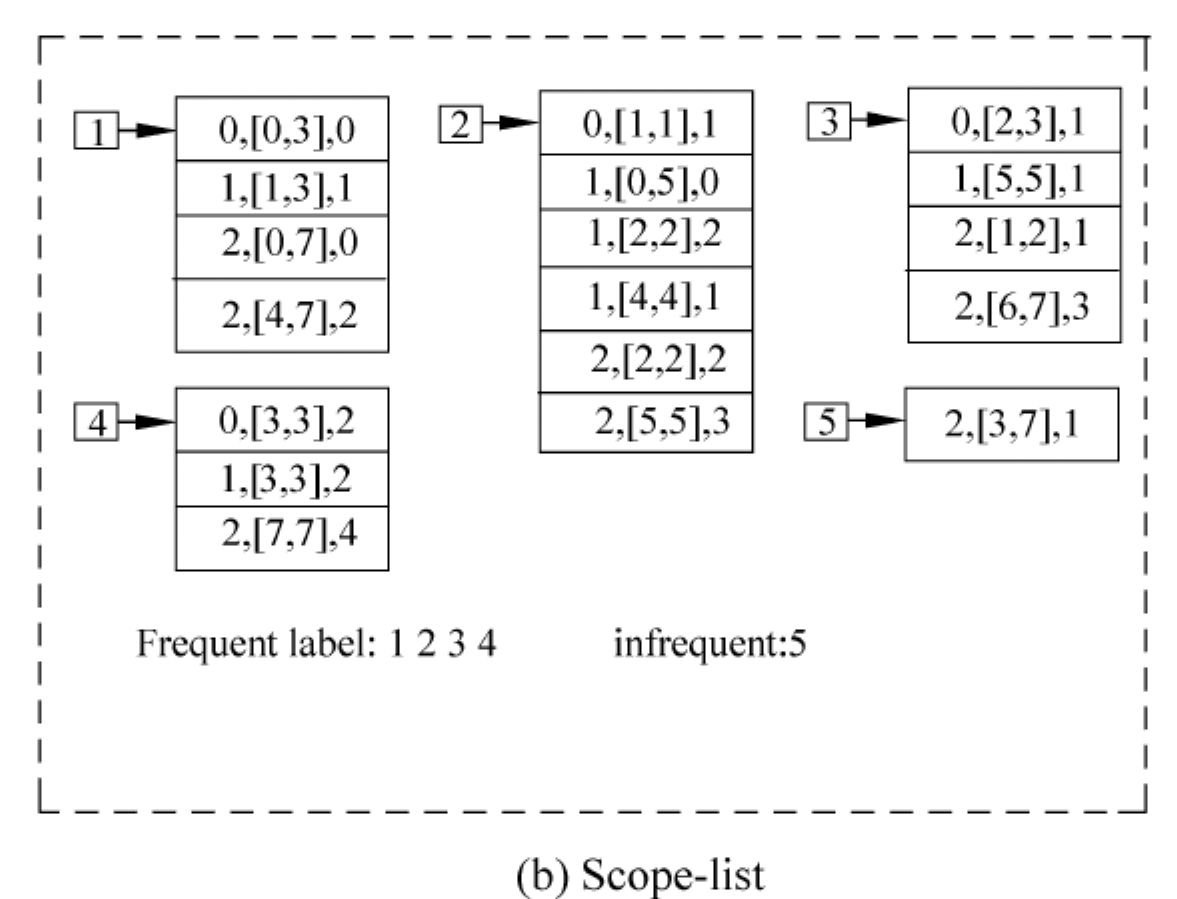
图 4-1(a)所示数据库 D 有三个文档,用户最小支持度为 minsup=70%(为了简化问题,最小支持度取值较大,一般情况下取 0.5),挖掘频繁子树过程如下:

第一步,通过扫描数据库(遍历文档树)得到每个结点的三元编码,具有相同标签的结点构成邻接表,称为范围列表(scope-list),采用图 4-1(b)的数据结构进行存储,$L=\{1,2,3,4,5\}$。通过范围列表很容易计算出结点标签 1、2、3、4 的支持度均为 100%,为频繁结点;标签 5 结点的支持度只有 1/3,小于 70%,不是频繁结点,故 $F1=\{1,2,3,4\}$,$P0=\{<1,-1>,<2,-1>,<3,-1>,<4,-1>\}$。(对应算法的 1～3 行)

第二步,递归调用 Enumerate-subtrees( )过程,由 k-Subtrees 构造(k+1)-Subtrees。以 2-Subtrees 为例,P0 中 4 个结点任意两个结点都有可能进行连接,构成含有两个结点的子树,所以算法中(第 6、8 行)对任意两对元素都要枚举,包括自身之间。但并不是任意两个元素都能构成子树,如(1,-1)和(4,-1)可以构成 2-Subtrees,反之则不能构成 2-Subtrees,因为标签 1 是 4 的祖先结点。为了减少范围列表的连接次数,算法通过结点的层次关系直接判断出子树的可能性(第 9 行),只有可能构成子树的结点才进行连接,排除了冗余计算,同时对构成的子树只有频繁子树才是所需结构,所以当 11 行条件满足时才进行类扩展,通过这两个条件的限制,大大减少了算法的时间开销。如构造 2-Subtrees 时,子树(2,1,-1),(3,2,-1)等子树支持度均小于 70%,不是频繁子树,类扩展时过滤掉。见图 4-2 中的




(a) 文档



(b) Scope-list

图 4-1　文档森林及结点数据结构

2-Subtrees 构造过程。

第三步，对每个频繁子树处理并存储其前缀编码 prefix 和拓扑编码 code（第 14、15 行）。

第四步，枚举调用 Enumerate-subtrees（[Px]）过程，直至挖掘出所有频繁子树，如图 4-2 中的 3-Subtrees 和 4-Subtrees。

图 4-2 给出了数据库 D 的全部频繁子树的挖掘过程，按照结点数和支持度大小顺序依次为：

S1：12-134-1-1，S2：12-14-1，S3：12-13-1，S4：134-1-1，S5：12-1，S6：14-1，S7：13-1，S8：34-1

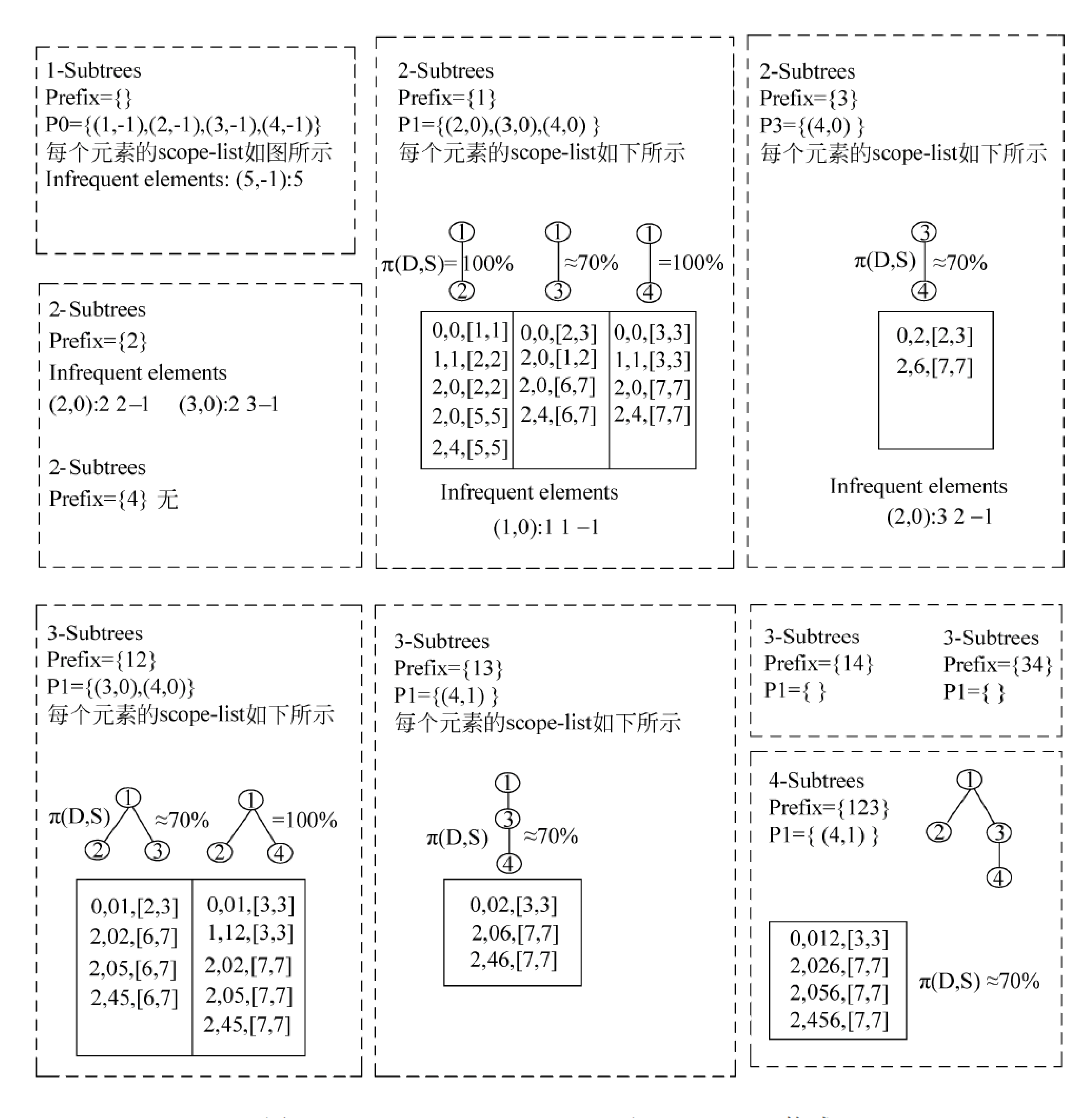


图 4-2　2-Subtrees、3-Subtrees 和 4-Subtrees 构成

4.2　文档结构的相似度计算

由于具有结构化、可扩展性、跨平台性等特点，越来越多的数据标准采用 XML，如 MathML、NewsML、OWL、LOGML、ebXML、cnXML 等，XML 成为信息存储和交换的主要形式，继而从 XML 文档结构中获取所需的模式成为 XML 应用领域的研究热点，如对 XML 文档结构挖掘可以为生物信息

学、网络日志分析、Web 结构分析和分类等提供重要知识，其中 XML 文档的结构相似度量是 XML 结构分析的基础核心问题。已有的 XML 文档结构相似性度量主要包括距离编辑法[21-22]、路径匹配法[23]和时序分析法[24]，基本上都是将 XML 文档视为一棵标记树，编辑距离的思想是把一个文档转化为另一个文档所需操作（更名、删除、插入）的最小代价作为度量标准，通过图匹配算法计算相似性，当文档很大时，计算代价太高；而路径匹配法基于边匹配，采用 Jaccard 系数来度量，但度量相似性的精度不高；时序分析法把 XML 文档结构表示为一个时间序列，每个标签的出现相应于一个脉冲，通过相应傅立叶变换来计算文档间的相似性，但是这种方法无法排除标签的多次重复出现对计算结构相似度的影响。

文献[20]提出了一种高效挖掘频繁子树的迭代算法 TreeMiner，能够在用户给定的最小支持度阈值内找出所有的频繁子树，文献[9]给出了一种挖掘无序频繁子树的高效算法。本节在 TreeMiner 算法基础上结合文本分类的向量空间模型（Vector Space Model）提出了 XML 文档频繁结构向量模型[7]（Frequent Structure Vector Model，FSVM），通过频繁结构单元在文档中出现的频度和权重定义了文档结构相似度。

4.2.1 频繁结构向量模型

文献[8]采用了向量空间模型来表示 XML 文档，提出结构链接向量模型，将 XML 文档中的每个结构单元看作一个向量，类似于 VSM 模型的一个文档，但没有就如何选择结构单元给出具体方法，只简单将每个结点作为基本结构单元，不能完备表示 XML 文档结构信息。本节将给出频繁结构向量模型 FSVM 的表示及 XML 文档结构相似性度量方法，应用 TreeMiner 算法挖掘的频繁子树表示结构向量空间中的结构单元。

将每个 XML 文档看作一棵带有标签的有序树 $T=(r,V,B,L)$，其中 r 表示根结点，V 表示树中结点集合，B 表示边的集合，L 表示所有结点标签的集合。那么 XML 文档集合 D（又称数据库）视为有序树的集合，$D=\{t0,t1,t2,\cdots\}$，称为文档森林，其中每棵树有唯一的标识 tid，$tid\in\{0,1,2,\cdots\}$，分

别表示树 $t0,t1,t2,\cdots$

定义 4-1　有序树(Ordered Tree)。XML 文档有序树 $T=(r,V,B,L)$ 中，$V=\{n_0,n_1,n_2,\cdots,n_{|T|-1}\}$，$nid\in\{0,1,2,\cdots\cdots\}$，按照深度优先先根序列编号，$|T|$表示树 T 的大小，即结点个数。结点不区分元素和属性，均采用圆圈表示，每个结点的标签用集合 $L=\{1,2,3,\cdots\cdots,m\}$表示，不同的结点可以有相同的标签。每个边 $b=<x,y>\in B$，是一个有序结点对，其中 x 是 y 的父结点。图 4-3 即为一棵有序树，其中 $r=n0$，$V=\{n0,n1,n2,n3,n4,n5,n6\}$，$B=\{<n0,n1>,<n1,n2>,<n2,n3>,<n2,n4>,<n1,n5,><n0,n6>\}$，$L=\{1,2,3,4\}$，$|T|=7$。

定义 4-2　树的拓扑编码表示(Topology code)。按照先根深度优先遍历树中所有结点的标签序列称为树的拓扑编码表示，用－1 表示从一个结点回溯到父结点的序列，图 4-3 所示有序树的拓扑编码为“1131-12-1-14-1-12-1”。

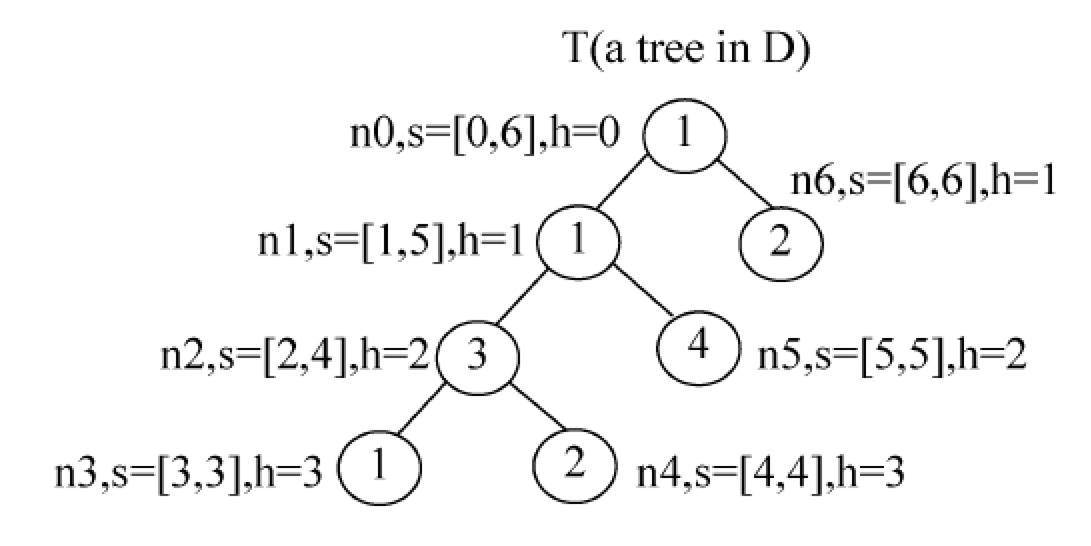


图 4-3　XML 文档树结点编码

定义 4-3　三元组编码(Triple code)。有序树中每个结点采用三元组(tid,scope,high)编码，$tid\in D$；scope 为结点的辖区范围，定义为[1,u]区间形式，1 为拓扑编码中本结点编号，u 为右子树最后结点的 nid，$high$ 表示每个结点在树中的层次。图 4-3 是树 T 中结点编码的实例，如根结点 $n0$ 的 nid 值为 0，右子树最后结点的 nid 值为 6，所以 scope 为 $s=[0,6]$，根结点的 $high=0$，其三元组编码为(n0,[0,6],0)。

定义 4-4　嵌入子树(Embodied Subtree)。树 $S=(r_s,V_s,B_s,L_s)$和 $T=(r_t,V_t,S_t,L_t)$，如果同时满足以下条件：i) $L_s\subseteq L_t$；ii)对于任意边 $b=(x,y)\in Bs$，在 T 中当且仅当 x 是 y 的祖先，即 T 中存在从 x 到 y 的路径，称为分

支，称 S 为 T 的嵌入子树，用 $S \leqslant T$ 表示。图 4-4 中，$S1$、$S2$ 均为树 T 的嵌入子树。这里需要强调的是在树 T 中 x 不一定是 y 的父结点，只要是祖先结点即可，这是与传统子树定义的区别，正是这一点，保证了在大型文档树中挖掘频繁子树的完备性。

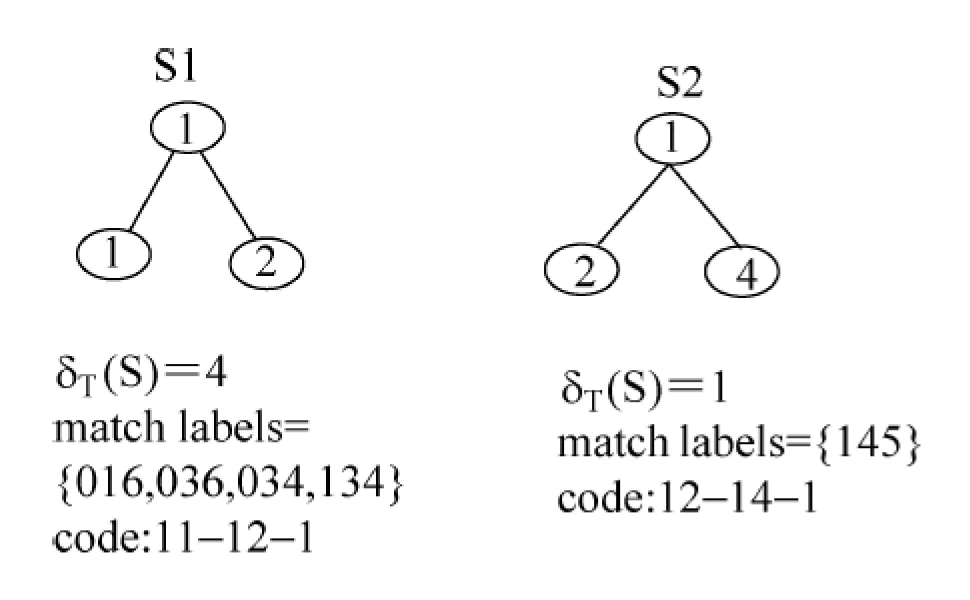


图 4-4 嵌入子树支持度及匹配标签

定义 4-5 子树的支持度(Support of Subtree)。嵌入子树 S 在树 T 中出现的次数，称为树 T 对子树 S 的支持度，以 $\delta_T(S)$表示，如 $\delta_T(S1)=4$，$\delta_T(S2)=1$。数据库 D 对子树 S 的支持度定义为：

$$\pi(D,S) = \sum_{T \in D} dT(s) / |D|, \quad 其中：dT(s) = \begin{cases} 1 & 当\ \delta_T(S) > 0 \\ 0 & 当\ \delta_T(S) = 0 \end{cases} \tag{4-1}$$

$|D|$表示数据库中有序树的总数。

定义 4-6 频繁子树(Frequent Subtree)。如果数据库 D 对嵌入子树 S 的支持度 $\pi(D,S)$大于或等于用户给定的最小阈值 minsup，则称子树 S 为频繁子树。

定义 4-7 匹配标签(Match Labels)。频繁子树 S 中所有结点对应树 T 相应结点的标签序列称为 S 的匹配标签。更一般的定义：设 D 为文档集合对应的森林，$\{n_1, n_2, \cdots, n_n\}$是 T 中的结点，$T \in D$，$\{s_1, s_2, \cdots, s_m\}$是 T 的频繁子树 S 的结点集合，则 S 的匹配标签为$(n_{i1}, n_{i2}, \cdots, n_{im})$，其中：

(1) $L(S_k) = L(n_{ik}), k=1,2,\cdots,m$ ；

(2) 对于 S 中每个边(S_j, S_k)，对应树 T 中的分支(n_{ij}, n_{ik})。

如图 4-4 中 $S1$ 和 $S2$ 的 match labels。

4.2.2　XML 文档的结构向量表示

向量空间模型以特征词构造一个高维空间，每个词语为该空间的一个维，文档被看作这个空间的一个向量，$d_x=(d_{w1},d_{w2},\cdots\cdots d_{wn})^T$，其中，$n$ 是文档集合中不同词语的个数。TFIDF（Term Frequency Inverse Document Frequency）是向量空间模型中一种常用的文档向量化方法，它综合考虑了词语在单个文档中出现的频度和该词语在文档集合中出现的频度，$d_{wi}=TF(w_i,d_x)*IDF(w_i)$。其中，$TF(w_i,d_x)$ 是词 w_i 在文档 d_x 中出现的次数；$IDF(w_i)=\log(|D|/DF(w_i))$，$|D|$ 是文档集合中文档总数，$DF(w_i)$ 是包含词语 w_i 的文档个数；$IDF(w_i)$ 是词语 w_i 的全局特性，用来体现词语 w_i 区分文档的能力。

向量空间模型能够有效描述文档内容，但不能体现文档的结构信息，而 XML 文档中含有丰富的结构信息，这些信息就蕴藏在文档树的频繁子集中。基于此，我们对传统向量空间模型进行了演变，以文档集的所有频繁子树构造一个结构特征空间，每个频繁子树为该空间的一个维，文档（树）d_t 被看作这个结构空间的一个向量，$d_t=(d_{s1},d_{s2},\cdots\cdots d_{sn})^T$，其中 s_i 是文档集合中不同的频繁子树，d_{si} 表示频繁结构 s_i 在文档树 d_t 中的权重，$d_{si}=TF(s_i,d_t)*IDF(s_i)$，其中 $TF(s_i,d_t)=\delta_t(s_i)*B(s_i)$，$B(s_i)$ 为频繁子树 s_i 的边数；$IDF(s_i)$ 是结构 s_i 的全局特性，数值越小，说明该结构在文档树集合中的区分能力越强，定义为 $\log(|D|/DF(s_i))$，$DF(s_i)$ 是含有子树 s_i 结构的树的个数，为了使每个频繁结构有作用，修订为 $\log(|D|/DF(s_i)+0.5)$。这样得到每个频繁子树的权重函数为：

$$d_{si}=\delta_t(s_i)*B(s_i)*\log(|D|/DF(s_i)+0.5) \tag{4-2}$$

这样，每个 XML 文档结构表示为一个向量 dt，整个数据库集合表示为结构—文档关系矩阵（Structure-Document Matrix）A_{nm}。其中，m 为频繁结构个数，n 为文档个数，矩阵中元素 $A(i,j)$ 表示第 j 个频繁结构在第 i 个文档中的权重，如表 4-1 所示。

表 4-1 XML 结构特征—文档关系矩阵

权重	S1	S2	……	Sm
文档 1($t1$)	A_{11}	A_{12}	……	A_{1m}
文档 2($t2$)	A_{21}	A_{22}	……	A_{2m}
……	……	……	……	……
文档 n(tn)	A_{n1}	A_{n2}	……	A_{nm}

4.2.3 文档相似性度量

与 VSM 模型类似，在 FSVM 模型中，同样采用频繁结构向量夹角的余弦来度量文档结构的相似性：

$$Sim(T_x, T_y) = \cos(T_x, T_y)$$

$$= \sum_{i=1}^{n}(\mathrm{d}x(si) * \mathrm{d}y(si)) \Big/ \sqrt{\sum_{i=1}^{n}\mathrm{d}x^2(si) * \sum_{i=1}^{n}\mathrm{d}y^2(si)} \quad (4\text{-}3)$$

其中，Tx，Ty 表示两个文档树，$\mathrm{d}x(si)$和 $\mathrm{d}y(si)$分别表示频繁结构 si 在文档树 Tx、Ty 中的权重。从式(4-3)可以看出，计算未知文档与已知文档集合中某个文档的相似度问题归结为以下步骤：

第一步：在文档集合中挖掘所有频繁子树，根据定义 4-6 和公式(4-1)判断频繁子树；

第二步：对拓扑编码表示的频繁子树排序，应用定义 4-5 和公式(4-2)计算权重，构造结构特征空间，生成结构—文档关系矩阵；

第三步：计算未知文档在结构特征空间的向量表示；

第四步：应用公式(4-3)计算文档结构相似性。

两个文档结构向量的内积越大，即 $Sim(Tx, Ty)$值越大，说明相似度越大，如果按结构分类分在同一类的可能性越大，反之可能性越小。这里的关键是第一步挖掘频繁子树问题。

4.2.4 计算实例

假定图 4-1 数据库 D 为已知文档集合，$|D|=3$，文档树分别为 $t0$、$t1$ 和

$t2$，利用改进的 TreeMiner 算法挖掘出 8 个频繁子树 $S1$-$S8$。图 4-3 为未知文档树，标记为 t，根据定义 4-5，计算频繁子树的支持度 $\delta_t(s_i)$ 即频度，同时计算出频繁子树中含有的边数及局部特征参数 $DF(s_i)$，结果如表 4-2 所示。

表 4-2　频繁子树在每个文档中的频度及相关参数

频度	$S1$	$S2$	$S3$	$S4$	$S5$	$S6$	$S7$	$S8$
$t0$	1	1	1	1	1	1	1	1
$t1$	0	1	0	0	1	1	0	0
$t2$	3	3	3	2	3	2	3	1
t	0	1	2	0	3	2	2	0
$B(si)$	3	2	2	2	1	1	1	1
$DF(si)$	2	3	2	2	3	3	2	2

根据权重函数公式(4-2)计算出每个频繁结构的权重，结果如表 4-3 所示。

表 4-3　频繁子树在 FSVM 中的向量表示

权重	$S1$	$S2$	$S3$	$S4$	$S5$	$S6$	$S7$	$S8$
$t0$	0.9031	0.3522	0.6021	0.6021	0.1761	0.1761	0.1761	0.1761
$t1$	0.0000	0.3522	0.0000	0.0000	0.1761	0.1761	0.0000	0.0000
$t2$	2.7093	1.0565	1.8062	1.2041	0.5283	0.3522	0.5283	0.1761
t	0.0000	0.3522	1.2041	0.0000	0.5283	0.3522	0.3522	0.0000

根据 XML 文档结构相似度函数公式(4-3)计算任意两个文档之间的相似度为：

$sim(t0,t1)=0.322521$，　$sim(t0,t2)=0.985432$，　$sim(t1,t2)=0.328011$

$sim(t,t0)=0.549918$，　$sim(t,t1)=0.446359$，　$sim(t,t2)=0.574932$

根据数据结果表明，未知文档 t 与文档 $t2$ 的结构最相似，同时文档 $t0$ 与文档 $t2$ 的结构最相似，这在文档树图示中表现十分明显。可见基于频繁结构向量模型的 XML 文档相似度量方法有很高的计算精度。

4.3　基于结构和内容联合提取的 XML 文档相似度量

XML 文档富含结构信息，但同时也具有文本内容，所以除了考虑其结构信息外，还需分析其基于关键词特征提取的语义内容信息，特别是如何将

二者有机联合进行特征提取和表示，是本节解决的根本问题。

4.3.1 XML 文档模型及特征分析

XML 文档是文本文件，均为 Unicode 编码，基本内容包括 XML 声明(Declaration)、注释(Comment)、标记(Tag)、元素(Element)、属性(Attribute)和内容(Text)几个部分组成，前两部分主要表明文件的版本信息和用途等内容，数据部分体现在后四部分，是我们主要研究对象。一个有效的 XML 文档被视为一棵带标签的结点树，称为文档模型，为了简化问题，不区分元素和属性结点。图 4-5 所示为杂志和会议论文集中学术论文的 XML 文档结构模型。

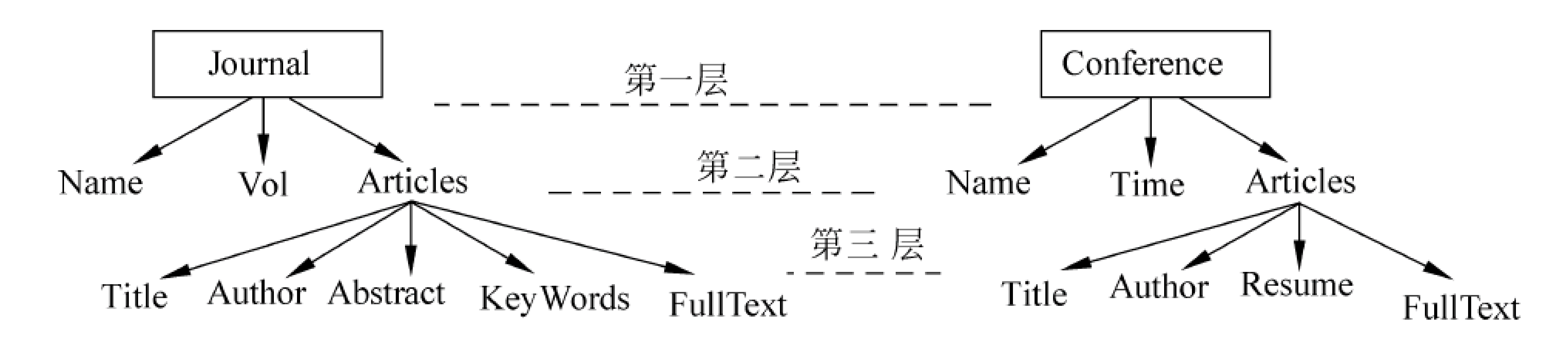


图 4-5 杂志和会议论文集 XML 文档结构模型

一个 XML 文档对应一棵有向树 $T=<V,E>$，其中 V 由文档中所有标签元素结点或者标签的属性构成，对于集合 E 定义为：若元素 $a\in V$，且元素的子元素或属性 $b\in V$，则边$(a,b)\in E$。

从图中可以看出，XML 文档中元素和子元素构成了文件的分层结构，位于不同层次的结点标记具有不同的重要性，据此分类时应有不同的权重分配，文档中的数据关键词信息基本出现在文档树中叶结点上，因此根据内容分类时可以只考虑叶结点，标签分支结点更多是结构信息。这样文档树中结点标签和叶结点关键词就构成了文档的内容信息特征。

4.3.2 频繁结构层次向量模型

XRules 分类只考虑 XML 文档结构信息，通过挖掘满足一定支持度的结构规则进行分类，适用于 XML 结构信息利用，如生物信息学等领域。很

显然，除了结构影响 XML 的分类外，文档内容中的关键词对分类也起着重要的作用，如两篇文献资料按照学术研究领域分类既要考虑其文档结构，同时更要处理其中描述的内容，为此，在提出的频繁结构向量模型中融合语义内容信息，即除了考虑文档结构信息外，同时考虑结构中关键词内容信息，提出了频繁结构层次向量模型作为 XML 网页分类的数据模型[25-26]。

该模型将一个文档从结构上划分为 N 层，对频繁结构中基于每层的文本段内容（叶结点）建立相应的文本特征向量以及文本层次权值向量。这样将每个频繁结构看作一个行向量，类似于 N 层 VSM 中的文本段，每个 XML 文档则被量化为一组向量，以一个矩阵来表示，称为特征值矩阵，如表 4-4 所示，从而达到将半结构化文本的结构分析与文本内容分析相结合的目的，其中 $w(i,j)$ 表示第 j 个关键词在第 i 个频繁结构中的权重。

表 4-4　频繁结构层次向量的文档特征矩阵

文本层次特征 / 频繁结构特征	Key_1	Key_2	……	Key_n
S_1	$w(1,1)$	$w(1,2)$		$w(1,n)$
S_2	$w(2,1)$	$w(2,2)$		$w(2,n)$
……				
S_{km}	$w(m,1)$	$w(m,2)$		$w(m,n)$

4.3.3　XML 文档结构和内容联合相关度计算

频繁结构层次向量模型将 XML 文档看成是由文档结构和若干关键词的特征构成，每个文档被表示成一个 $m \times n$ 的矩阵，如表 4-4 所示，其中 m 为文档集合中不同的频繁结构（按照支持度降序排列），n 为文档集合中不同词语的个数。

XML 文档频繁结构的挖掘采用 TreeMiner$^+$ 算法，结构单元中的结点记载了源文档中的层次属性，这与文本内容的权重相关。图 4-6 中的频繁结构（假定 Minsup = 50%）中，结点集合{Journal，Name，Vol，Articles，Title，Author，Abstract，KeyWord，FullText，Confrence，Resume}均为频繁结点，即为 1-Subtrees，2-Subtrees 共有 18 棵，图 4-6 所示为部分子树。

每个频繁结构的权重函数定义采用公式(4-2)。

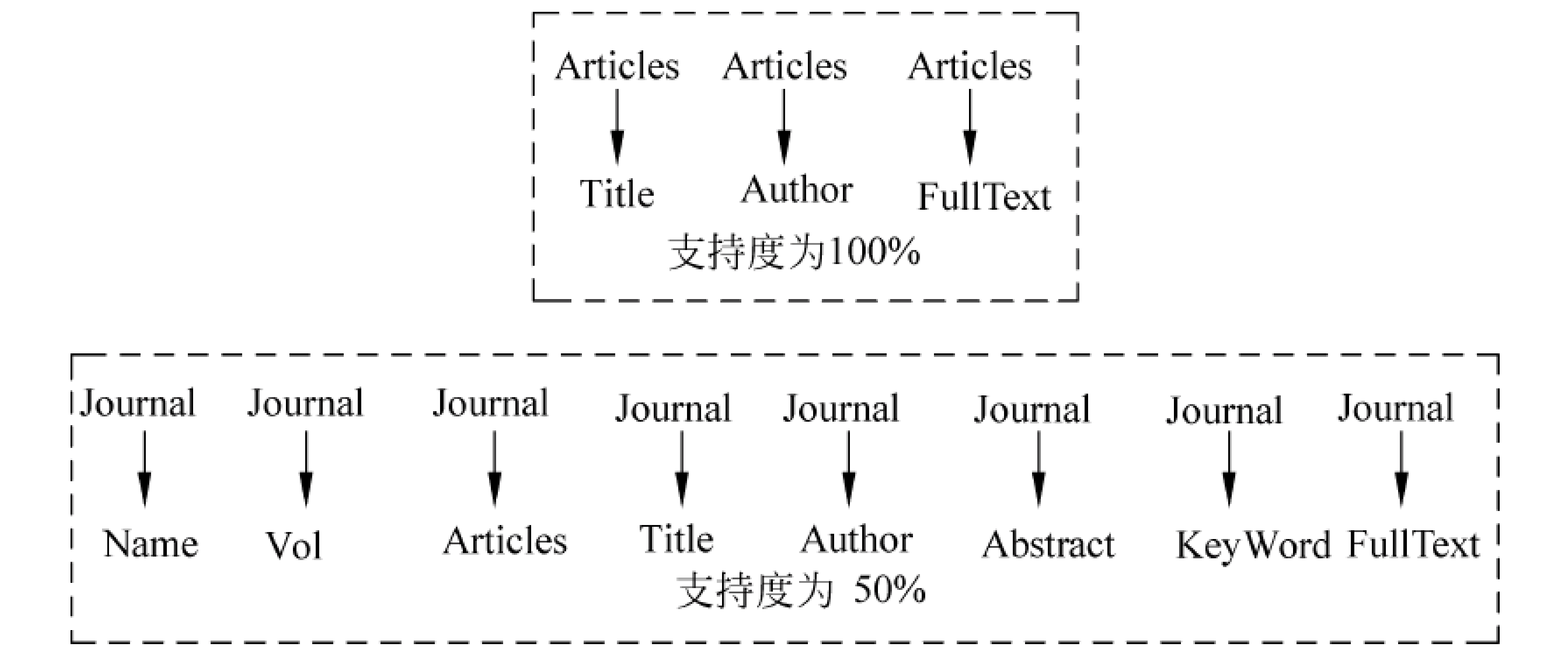


图 4-6 部分频繁子树 2-Subtrees

对每个文档模型树的叶结点按先序编号，如图 4-7 所示的文档树，分别统计计算各个叶结点关键词的频率 $TF_{k,d}^{i}=tf_{k}^{i}(d)$，其中 j 为叶结点序号。叶结点文本关键词在文档中的位置权值 $w_{k,d}^{i}$ = 父结点层权值/兄弟结点个数$\left(根结点层数为 1，其中 \sum w_{k,d}^{i}=1\right)$。如 Name 的位置权值为 1/3，Author 的位置权值为 1/9。这样 XML 文档 d 中第 k 关键词的频率表示为：

$$TF_{k,d}^{i}=\sum(w_{k,d}^{j}*TF_{k,d}^{j}) \tag{4-4}$$

式中 w 结合 Salton 提出的普通本文和 HTML 的权值计算方法[27]，修正为：

$$w_{i}=\frac{TF_{k,d}^{i}\log(N/n_{i})}{\sqrt{\sum_{j}(TF_{k,d}^{i}\log(N/n_{j}))^{2}}} \tag{4-5}$$

其中 N 为所有文档的数目，n_i 是出现该关键词的文本数。

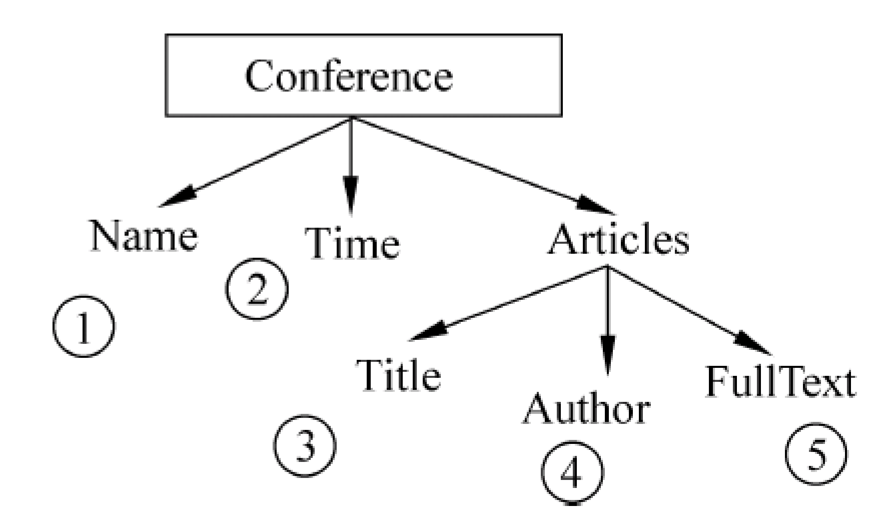


图 4-7 XML 文档树的叶节点编号

4.4　基于粗糙集理论的网页分类技术

在进行特征权重计算和特征过滤后，从而得到训练集 XML 文档的结构和内容特征值矩阵表示[28]，如表 4-5 所示，其中 $S=\{S1,S2,\cdots,Sm\}$，为文档数据库的所有频繁结构，Ds 表示每个频繁结构的权重，由公式(4-2)得到。$\{KW1,KW2,\cdots,KWn\}$为内容特征关键词，$TFi(Sj)$表示关键词 i 在结构 Sj 中出现的频率，由公式(4-4)和公式(4-5)得到。

表 4-5　XML 文档的频繁结构层次向量空间特征值

权重	Ds	$KW1$	$KW2$	……	KWn
S1	$ds1$	$TF1(s1)$	$TF2(s1)$	……	$TFn(s1)$
S2	$ds2$	$TF1(s2)$	$TF2(s2)$	……	$TFn(s2)$
……	……	……	……	……	……
Sm	dsm	$TF1(sm)$	$TF2(sm)$	……	$TFn(sm)$

ROSETTA[29]系统(罗塞塔)是挪威科技大学计算机与信息科学学院开发的在粗糙集理论的框架中进行数据分析的一个全面的软件系统，拥有一套灵活和强大的算法，并把这些算法放在一个用户友好的设计环境下旨在支持所有基于可分辨建模理论的情形，能够很好地进行属性约简、规则抽取、分类和绩效评价。我们即采用 ROSSETA 系统对测试数据集进行分类，图 4-8 为系统的详细处理流程。

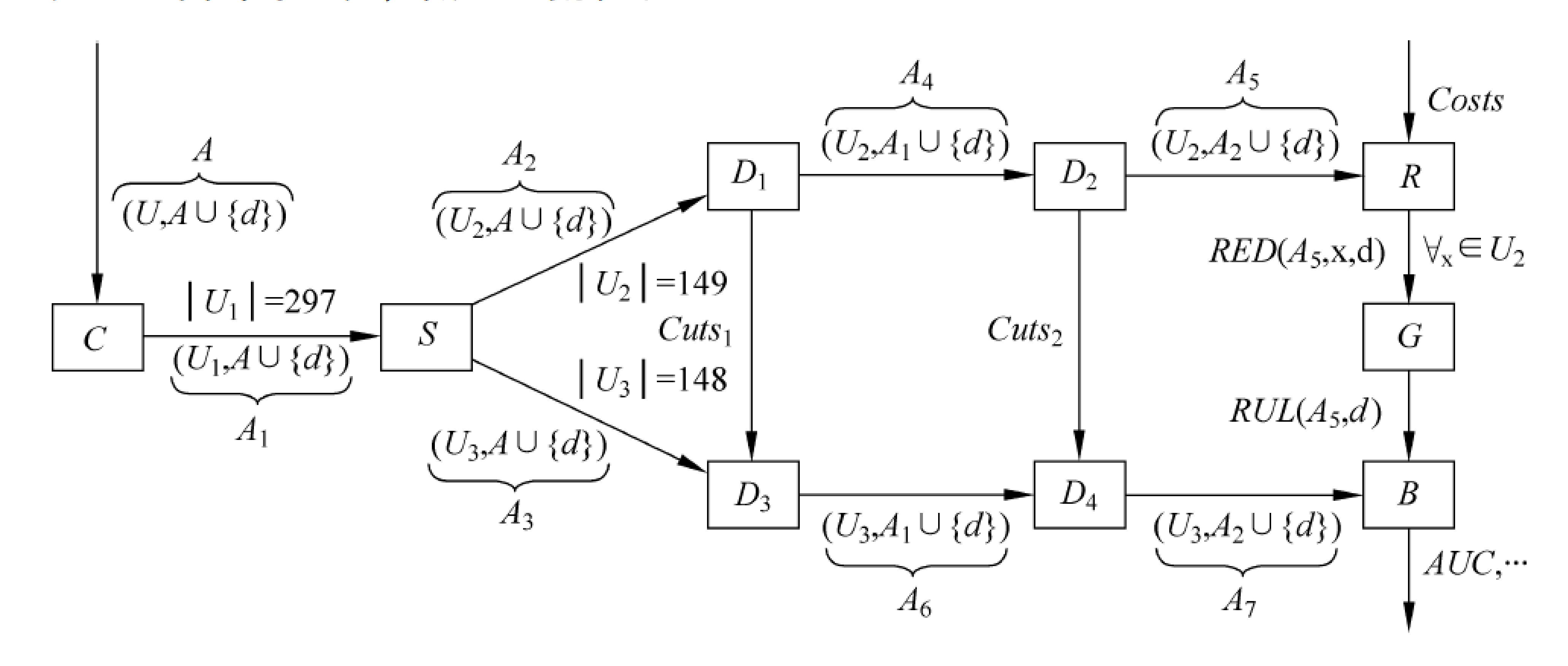


图 4-8　ROSSETA 系统机器学习结构

被输入的决策系统 A 的内容即为测试文档集合的特征值(表 4-5 中的数据)。A 第一次清除缺失值产生 A_1,然后在分裂成两个不相交的子系统,A_2 和 A_3。A_2 离散成一个两阶段程序,而 A_3 用由 A_2 计算产生的截集来进行离散。然后由经处理过 A_2 计算出约简和规则,并且把生成的规则用于对经处理过的 A_3 中的对象进行分类,最后从这个分类中获得绩效估计。

4.4.1 基于结构的分类

不考虑文档内容,只保留表 4-5 中频繁结构的特征值权重 Ds 列,整个 XML 文档集合的频繁结构构成特征空间,如表 4-6 所示,生成矩阵 A 的内容。

表 4-6 XML 文档结构特征关系矩阵

权重	$S1$	$S2$	……	Sm
文档 1($t1$)	A_{11}	A_{12}	……	A_{1m}
文档 2($t2$)	A_{21}	A_{22}	……	A_{2m}
……	……	……	……	……
文档 n(tn)	A_{n1}	A_{n2}	……	A_{nm}

4.4.2 基于内容的分类

不考虑文档结构,只保留表 4-5 中关键词的权重,整个 XML 文档集合的特征词构成特征空间,由公式(4-5)生成矩阵 A 的内容,如表 4-7 所示。

表 4-7 XML 内容特征—文档关系矩阵

权重	$KW1$	KW 2	……	KW m
文档 1($t1$)	A_{11}	A_{12}	……	A_{1m}
文档 2($t2$)	A_{21}	A_{22}	……	A_{2m}
……	……	……	……	……
文档 n(tn)	A_{n1}	A_{n2}	……	A_{nm}

4.4.3 基于结构和内容联合的分类

XML 文档分类有时受结构影响较大,如从发表论文是杂志论文还是会

议论文角度出发，图 4-5 中的文档 2 和图 4-7 中的文档属于一类，但有时从文章论述内容出发，关键词特征起重要作用，所以一个文档有时同属于多类。对结构和内容分别分配不同的权重可以联合提取其特征值，以达到不同分类目的。

每个频繁结构中的关键词特征事件是独立事件，每个频繁结构与在这个结构中出现的关键词之间是先验事件，表 4-5 中文档的每个关键词 KWi 在整个文档的权重 TFi 除了与其所在文档层次有关外，还与其所在频繁结构的权重有关，所以将其修订为公式(4-6)，解释为：文档 d 中第 i 个关键词的权重为此关键词在每个频繁结构的频次与频繁结构权重的乘积累加。

$$w_i = \sum TF_i(S_k) \times d_{sk} \quad k = 1,2,\cdots,m, \quad i = 1,2\cdots,n \qquad (4\text{-}6)$$

这样，每个文档的频繁结构层次向量空间由矩阵规范化为一个向量，整个 XML 文档集合的结构和内容特征规划为 n 个关键词的权重值矩阵，即为 A 的内容。也就是说，每个文档表示为形如表 4-8 中的矩阵，经公式(4-6)规范化为表 4-8 中的一个行向量，最终整个文档集合构成 XML 联合特征—文档关系矩阵，利用向量内积计算文档相似度。

表 4-8　XML 联合特征—文档关系矩阵

权重	$\sum TF_1(Sk) * d_{sk}$	$\sum TF_2(Sk) * d_{sk}$	……	$\sum TF_m(Sk) * d_{sk}$
文档 1($t1$)	A_{11}	A_{12}	……	A_{1m}
文档 2($t2$)	A_{21}	A_{22}	……	A_{2m}
……	……	……	……	……
文档 n(tn)	A_{n1}	A_{n2}	……	A_{nm}

4.4.4　实验结果及分析

实验使用 AMD 双核 Athlon 4000＋ 2.10GHz CPU，1G 内存的个人计算机，所用操作系统为 Windows 2000 Server，所有算法均使用 Java 语言实现，所用 JDK 为 Java 2 Platform Standard Edition 5.0 标准版。

1. 实验数据和实验结果

实验数据分为两类：一类采用 ACMSIGMOD[30] 数据集中的 OrdinaryIssuePage 和 IndexTermsPage XML 文件，用于基于文档结构的分类测试。数据子集使用情况见表 4-9，文档数量中的第一个数字作为频繁子树结构挖掘训练使用，后一个数字作为分类测试使用。ACMSIGMOD 文档中带有分类信息，每个文档通常属于多个分类，如果一个文档特征符合多个文档的分类规则，认为这个文档同属多类，实验中采用分类信息来替代人工标注。实验结果的准确率(precision)采用公式(4-7)进行评价。

$$P = (\text{分类正确的文档数})/(\text{测试文档总数}) \tag{4-7}$$

表 4-9　实验中使用的数据子集

数据集 Datasets	源数据 Sources	文 档 总 数
ACMSIGMOD-1	OrdinaryIssuePage(1999)	40＋20
ACMSIGMOD-2	OrdinaryIssuePage(2002)	20＋10
ACMSIGMOD-3	IndexTermsPage(1999)	40＋20

频繁结构的挖掘采用 TreeMiner+ 算法，文档集合的标签及编码采用人工标注，不同数据集合的分类测试结果如图 4-9 所示。

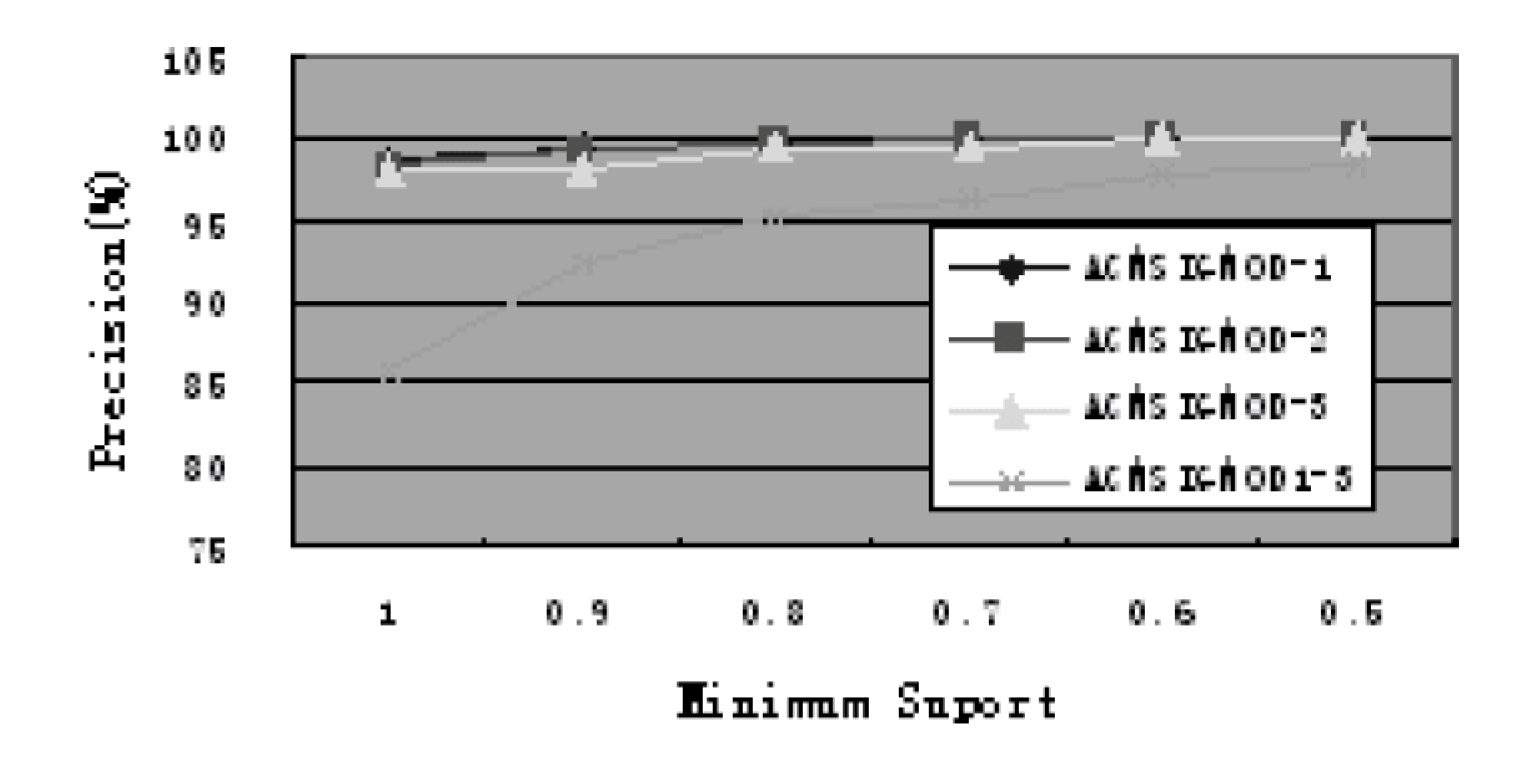


图 4-9　文档结构特征分类测试

另一类数据使用新浪网 RSS 频道聚合网站(http://rss.sins.com.cn)提供的基于 XML 格式的简易 RSS 新闻，每类新闻网页的主文档结构树如

图 4-10(a)所示，每项新闻结构如图 4-10(b)所示，为了说明结构和内容对分类的影响，我们有意对部分 XML 网页文件进行了一些结构修改，比如增删一些不太重要的元素(iamge、link、copyright，guid 等)，主要对其中 item 项进行结构和内容的联合特征提取测试。为了简化实验过程，内容特征的提取采用人工方法完成。

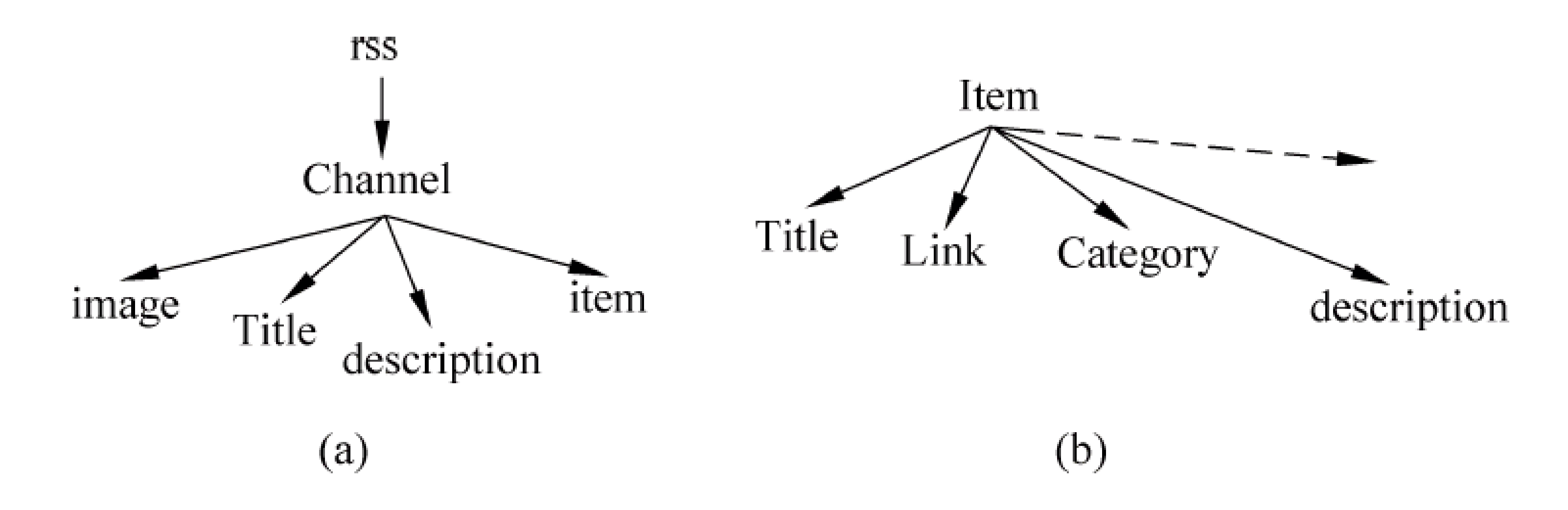


图 4-10　新闻网页的主要结构元素

共提取这样的 XML 网页 200 篇，其中 150 篇作为训练集，分为财经、军事、汽车、科技、体育 5 类，通过两种情况进行实验比较：①对于每个 XML 文档，仅考虑关键词特征频度；②对于每个 XML 文档关键词，考虑文档的频繁结构和关键词的位置权值和频度。获得特征值 A 后仍然使用 ROSSETA 系统进行训练和测试，查准率采用公式(4-7)表示，查全率(recall)采用公式(4-8)表示。实验数据及结果如表 4-10 所示，图示如图 4-11 和图 4-12 所示。

$$R = (\text{分类正确的文档数})/(\text{该类应有的文档数}) \tag{4-8}$$

表 4-10　基于 XML 文档结构和关键词特征分类实验结果

Dataset		Economic News	Military News	Automobile News	Technical News	Sports News
TrainingSamples(150)		30	30	30	30	30
TestingSamples(50)		10	10	10	10	10
Frequency structures(48)		12	8	9	11	8
Keywords(232)		62	44	25	65	36
Recall(%)	Only Keywords	75.6	76.8	85.2	72.3	85
	Considering Structure	91.5	93.1	95.8	92.3	96.2
Precision(%)	Only Keywords	83.2	86.5	87.4	84.6	87.6
	Considering Structure	95.6	97.2	97.6	96.3	98.1

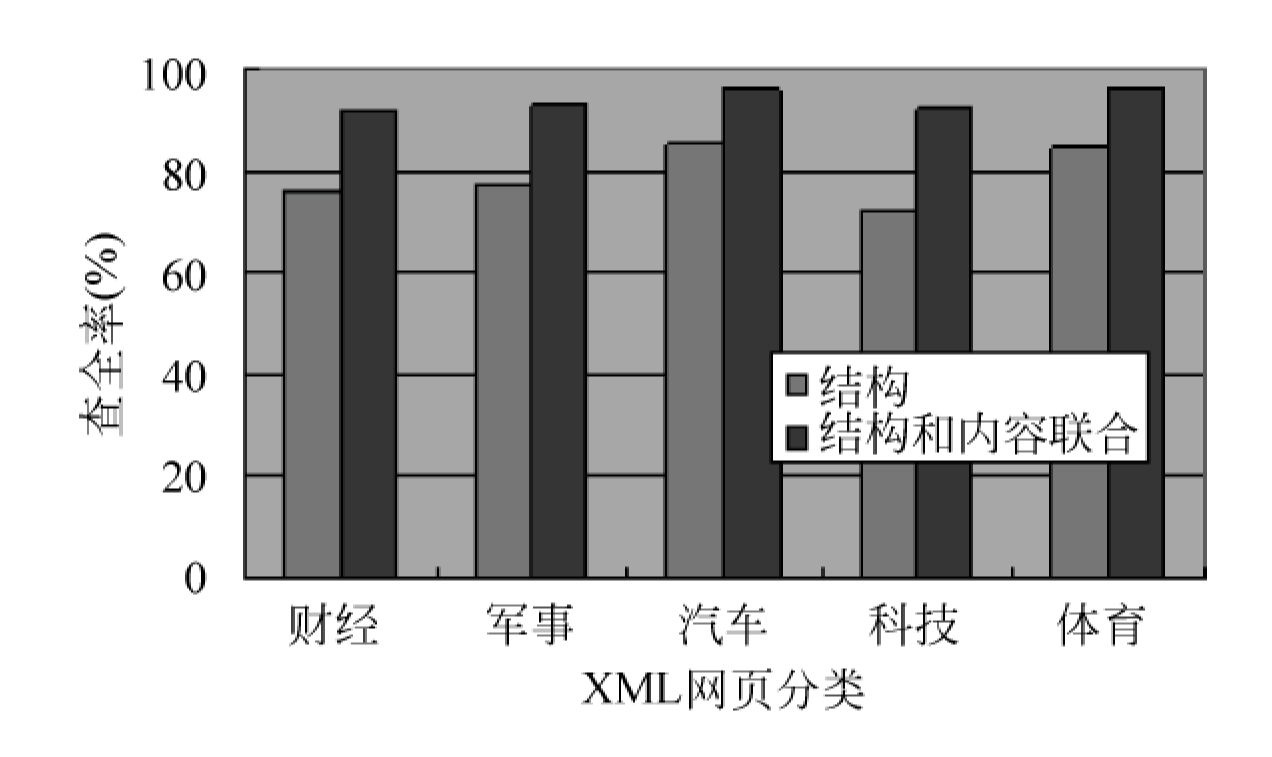


图 4-11　各类网页分类检索的查全率

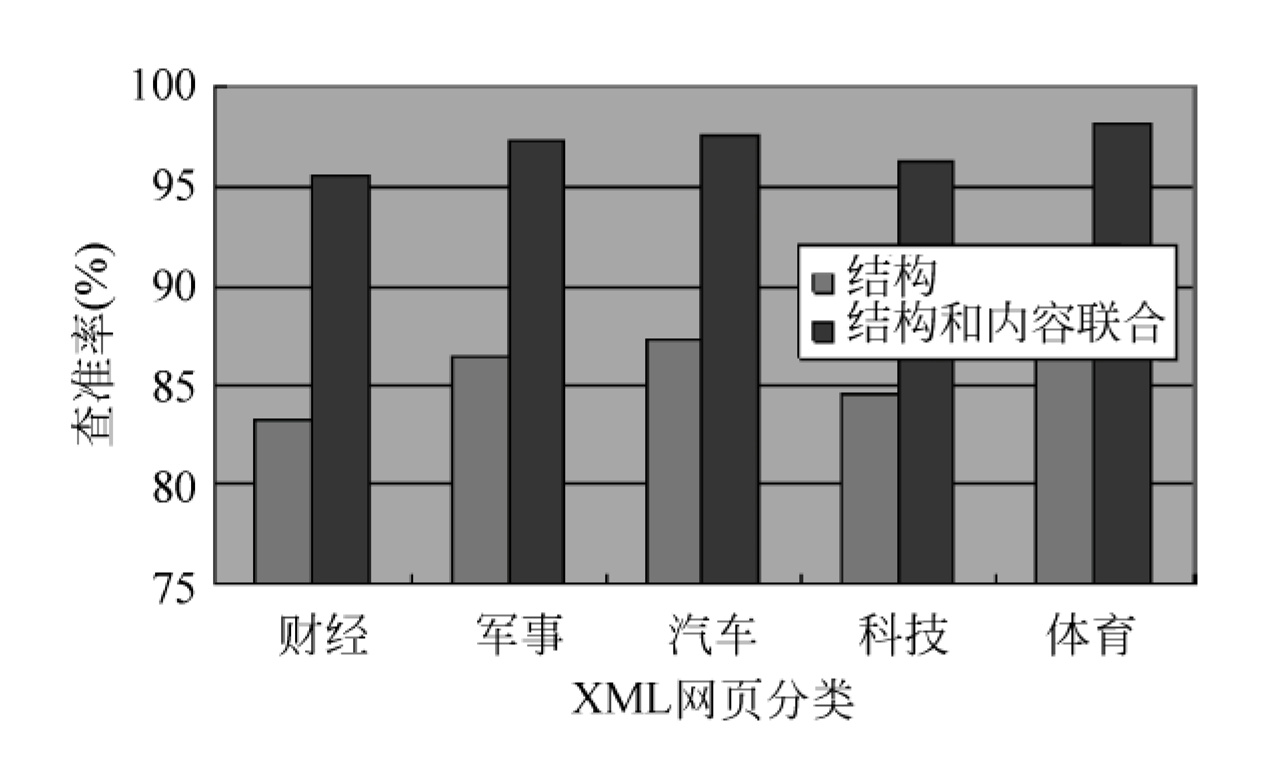


图 4-12　各类网页分类检索的查准率

2. 实验分析

第一组数据集主要针对 XML 文档结构分类，从图 4-9 可以看出，采取频繁结构向量模型作为 XML 文档结构分类有很高的准确率，特别对文献类文档，结构分类准确率可以达到 98％以上。

第二组数据主要针对文档内容和结构联合特征分类，从实验结果表 4-10 和图 4-11 及图 4-12 中可以看出，在 XML 网页分类中，只考虑文档内容，即关键词信息，查全率在 70％～85％，查准率在 85％左右；同时考虑文档内容和结构，联合提取结构和关键词特征值，分类和相似查询具有更高的查全率和查准率，可以达到 90％以上，特别体育新闻类网页查准率可以达到 98％。

实验数据表明，频繁结构层次向量模型对于 XML 网页的分类及实现相似查询具有很好的性能。

参考文献

[1] Yi J, Sundarcsan N. A classifier for semi-structured documents. In: Ramakrishnan R, Stolfo S, Pregibon D, eds. Proc. of the 6th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD 2000). New York: ACM Press, 2000. 340～344

[2] Denoyer L, Gallinari P. Bayesian network model for semi-structured document classification. Information Processing and Mangement, 2004, 40(5): 807～827

[3] Zhang ZP, Li R, Cao SL, Zhu YY. Similarity metric for XML documents. In: Ralph B, Martin S, eds. Proc. of the 2003 Workshop on Knowledge and Experience Management(FGWN2003). Karlsruhe, 2003. 255～261

[4] FLESCA S, MANCO G, MASCIARI E, et al. Detecting structural similarities between XML documents[A]. In Proc 5th Int. Workshop on the Web and Databases (WebDB'02) [C]. Madison, Wisconsin, 2002. 124～131

[5] M. J. Zaki. Effciently Mining Frequent Trees in a Forest. SIGKDD, 2002. 71～80

[6] M. J. Zaki, C C Aggarwal. Xrules: An effective structural classification for XML data [C]. Int'l Conf on Knowledge Discovery and Data Mining (SIGKDD'03), Washington, DC, 2003. 316～325

[7] 阎红灿，王淑芬等. 基于 TreeMiner 算法的 XML 文档结构相似度量方法[J]. 计算机应用研究. 2009，26(5): 1706-1709

[8] 杨建武，陈晓欧. 基于核矩阵学习的 XML 文档相似度量方法. 软件学报[J]. May 2006, 17(5): 991～1000

[9] 马海兵，王兰成. 高效挖掘无序频繁子树. 小型微型计算机系统[J]. Nov 2006，27 (11): 2104～2108

[10] 张冉，卡米力・毛依丁. 基于 XML 和 N 层 VSM 的 Web 信息检索[J]，计算机技术与发展，May 2006, 16(5): 56～58

[11] 牛强，王志晓，陈岱等. 基于 SVM 的中文网页分类方法的研究[J]. 计算机工程与设计，Apr. 2007, 28(8): 1893～1895

[12] 袁家政，须德等. 基于结构与文本关键词相关度的 XML 网页分类研究[J]. 计算机研究与发展, 2006, 43(8): 1361～1367

[13] 杨彦闯，杨炳儒等. 基于联合提取特征的粗糙集文本分类技术研究[J]. 计算机应用研究，2007，24 (7): 97～99

[14] 唐凯. 基于内容和分层结构的 XML 文件自动分类方法[J]. 计算机工程与应用，2007, 43(3): 168～172

[15] 韩景倜，卢致杰，覃正. 基于 XML 的复杂信息系统自动分类方法. 系统工程理论与应用，Dec. 2005, 14 (6): 488～492

[16] 瞿彬彬，卢炎生. 基于粗糙集的快速属性约简算法研究[J]. 计算机工程，2007，33 (11)：7～10
[17] 王效岳，白如江. 基于变精度粗糙集模型的 Web 文档分类方法[J]. 现代图书情报技术，2005. 12，131(12)：51～54
[18] 陈万领，陈卓宁，宾鸿赞. 基于粗糙集理论的 Web 工艺网页智能搜索. 武汉理工大学学报. Nov. 2005，27 (11)：76～79
[19] 李滔，王俊普，徐杨. 一种基于粗糙集的网页分类方法[J]. 小型微型计算机系统. Mar. 2003，24 (3)：521～524
[20] M. J. Zaki. Effciently Mining Frequent Trees in a Forest. SIGKDD，2002. 71～80
[21] CHAWATHES，RAJARAMANA，GARCIA-MOLINAH，et al. Change detection in hierarchically structured information[A]. In Proc ACM SIGMOD Int. Conf. On Management of Data(SIGMOD'96) [C] Montreal，Quebec，June 1996. 493～504
[22] Zhang ZP，Li R，Cao SL，Zhu YY. Similarity metric for XML documents. In：Ralph B，Martin S，eds. Proc. of the 2003 Workshop on Knowledge and Experience Management(FGWN2003). Karlsruhe，2003. 255～261
[23] COSTA G，MANCO G，ORTALE R，et al. A Tree-based Approach to Clustering XML Documents by Structure[Z]. Rappoito Tecnico N. 04：RT-ICAR-CS-04-04 April 2004. 137～148
[24] FLESCA S，MANCO G，MASCIARI E，et al. Detecting structural similarities between XML documents [A]. In Proc 5th Int. Workshop on the Web and Databases(WebDB'02) [C]. Madison，Wisconsin，2002. 124～131
[25] YAN Hongcan，JIN Dianchuan，etc. Feature Matrix Extraction and Classification of XML Pages，APweb 2008，LNCS4977，210-219
[26] YAN Hongcan，LI Minqiang，etc. An Efficient Structure Similarity Measure Method for XML documents based on Vector Space Model，DCABES 2008 Proceedings，345-353
[27] G Salton. Introduction to Modem Information RetrievalM]. New York：McGraw Hill Book Company，1983. 124～156
[28] 阎红灿，李敏强. 结构和内容联合提取的 XML 网页分类研究[J]. 天津大学学报. 2009，11(3)：272-276
[29] Aleksander Øhrn. Discernibility and Rough Sets in Medicine：Tools and Applications. Department of Computer and Information Science Norwegian University of Science and Technology. N-7491 Trondheim，Norway. http://www.idi.ntnu.no/～aleks/thesis/extabst.pdf
[30] http://www.acm.org/sigs/sigmod/record/XMLSigmodRecordMarch1999.zip http://www.acm.org/sigs/sigmod/record/XMLSigmodRecordNov2002.zip

第 5 章　基于 N-VSM 的全文检索

信息检索(Information Retrieval/Access)是从任何信息集合中识别和获取所需信息的过程及其所采取的一系列方法和策略。从原理上看,包括存储与检索两方面。知识是有组织的大量的信息,获得知识有赖于获得信息。本章将介绍非结构化数据的全文检索技术。

5.1　全文检索的关键技术

随着互联网和电子商务的发展,电子信息已经进入大数据时代,这些信息大致可分为两类:结构化数据和非结构化数据,结构化数据指的是诸如企业财务账目和生产数据、学生的分数数据等等,非结构化数据的则是一些文本数据、图像声音等多媒体数据等等。据统计,非结构化数据占有整个信息量的 80%以上。对于结构化数据,用 RDBMS(关系数据库管理系统)技术来管理是目前最好的一种方式。但是由于 RDBMS 自身底层结构的缘故使得它管理大量非结构化数据显得有些先天不足,特别是查询这些海量非结构化数据的速度较慢。而通过全文检索技术就能高效地管理这些非结构化数据。

经过多年的发展,全文检索从最初的字符串匹配程序已经演进到能对超大文本、语音、图像、活动影像等非结构化数据进行综合管理的大型软件。由于内涵和外延的深刻变化,全文检索系统已成为新一代管理信息系统的代名词,衡量全文检索系统的基本指标也逐渐形成规范。

全文检索是一种将文件中所有文本与检索项匹配的文字资料检索方法。全文检索系统是按照全文检索理论建立起来的用于提供全文检索服务的软件系统,系统的核心则具有建立索引、处理查询返回结果集、增加索引、

优化索引结构等功能。衡量全文检索系统性能的基本指标有查全率、查准率、响应时间和输出形式等。全文检索的中心环节是文件内容表达、信息查询的获得以及相关信息的匹配。一个好的全文信息检索系统不仅要求将输出信息进行相关性排列,还应该能够根据用户的意图、兴趣和特点自适应和智能化地调整匹配机制,获得用户满意的检索输出。

要实现全文检索,首先必须对 Web 信息进行预处理。

信息预处理的主要功能是过滤文件系统信息,为文件系统的表达提供一种满意的索引输出。其基本目的是为了获取最优的索引记录,使用户能很容易地检索到所需信息。

(1) 格式过滤:信息预处理应该能够过滤不同格式的文档,以及图片、声音、视频等信息。这使得搜索引擎不仅能够检索文字,而且能够检索原始格式文件的所有信息。

(2) 语词切分:语词是信息表达的最小单位,而汉语不同于西方语言,其句子的语词间没有分隔符因此需要进行语词切分。常用的语词切分方法有按词典进行最大词组匹配、逆向最大词组匹配、最佳匹配法,联想—回溯法、全自动词典切词等。近年来,又出现了基于神经元网络的和专家系统的分词方法和基于统计和频度分析的分词方法。

(3) 词法分析:汉语语词切分中存在切分歧义,如句子"网球拍卖完了",可以切分为"网球/拍卖完了",也可以切分为"网球拍/卖完了"。因此需要利用各种上下文知识解决语词切分歧义。此外,还需要对语词进行词法分析,识别出各个语词的词干,以便根据词干建立信息索引。对于英语语词,建立索引之前首先要去除一些停顿词(如常见的功能词 a、the、it 等)和词根(如 ing、ed、ly 等)。

(4) 词性标注和短语识别:在切分的基础上,利用基于规则和统计的方法进行词性标注。在此基础上,还要利用各种语法规则,识别出重要的短语结构。

(5) 自动标引:从网页文档中提取出一组能最大程度上概括其内容特征、可作为用户检索入口的关键性信息,用该组信息对文件进行标引,使用户可以通过输入关键信息检索到该文文件的简要信息,如标题、摘要、时间、作者和 URL 等,进一步点击可查询到该文档。

（6）自动分类：建立并维护一套完整的分类目录体系，根据文件的信息特征，计算出与其相关程度最大的一个或多个分类，将文档划归到这些分类中去，使用户可以通过浏览分类体系直接查询到该文档。

目前，搜索引擎的使用已成为排在收发电子邮件之后的第二大互联网应用技术。搜索引擎起源于传统的信息全文检索理论，即计算机程序通过扫描每一篇文章中的每一个词，建立以词为单位的到排文件，检索程序根据检索词在每一篇文章中出现的频率和每一个检索词在一篇文章中出现的概率，对包含这些检索词的文章进行排序，最后输出排序的结果。全文检索技术是搜索引擎的核心支撑技术。全文检索系统架构如图 5-1 所示。

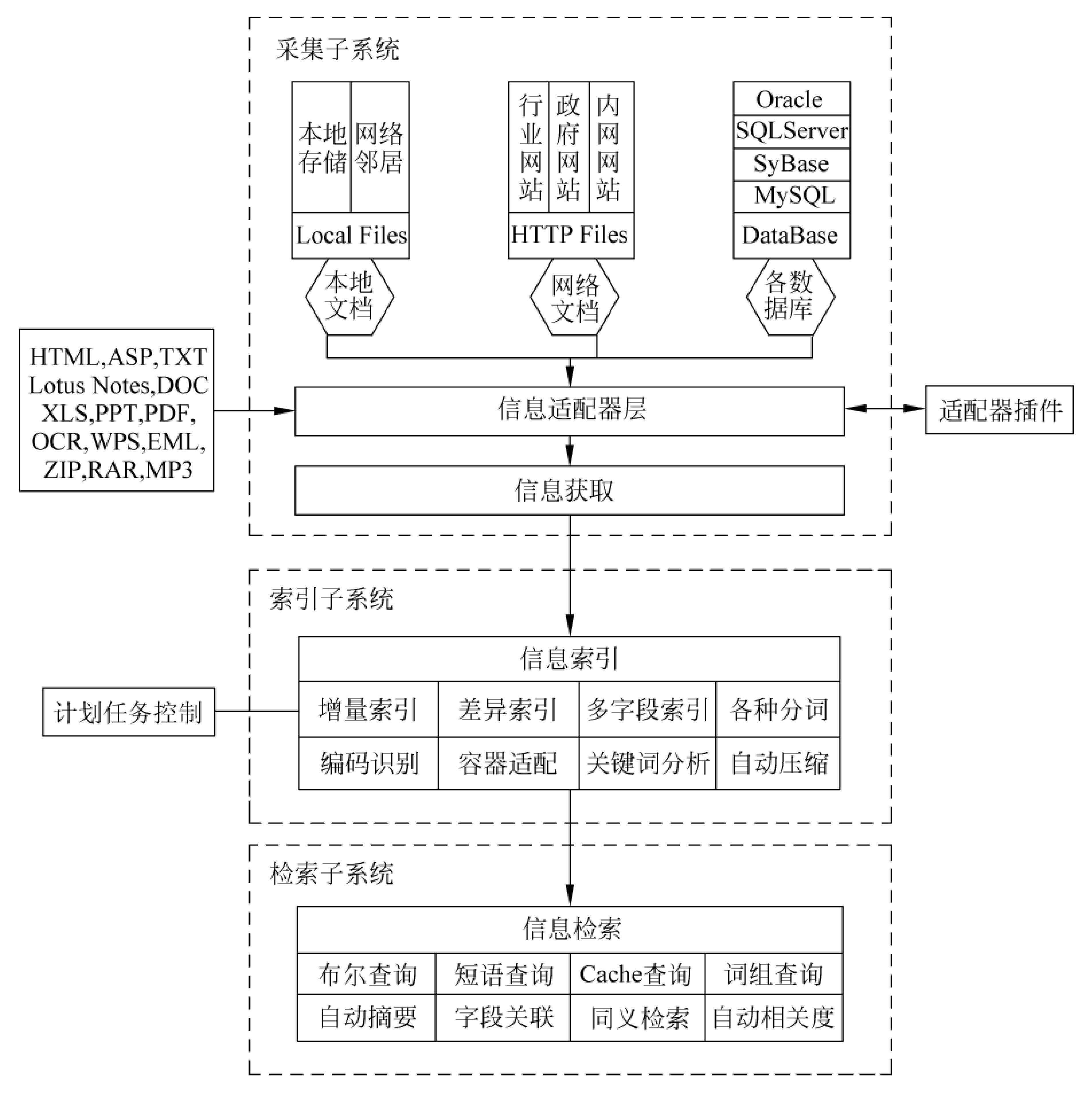


图 5-1　全文检索结构图

5.2 信息检索模型

实现全文检索的关键是解决检索模型的问题，即文档如何表示，如何建立索引项和文档之间的联系及检索结果如何处理等问题。原始的文档中包含文本、图像、视频、音频等数据，不能直接进行检索，需要从这些原始数据中抽取逻辑视图，以支持信息检索。用户用查询来表示信息需求，检索系统根据查询的表示，搜索文档集，获取与用户查询相关的文档。信息检索的匹配是相似性匹配，查询的结果按序返回。以上过程实际上涉及到三个重要的处理：文档集的逻辑表示、查询的表示、相似性匹配及其排序。也就是说，文档集、查询和相似性匹配决定一种检索策略和模式。对这些检索的因素和过程建模，就产生了各种不同的信息检索模型。信息检索模型的定义如下：

一个信息检索模型是将文档表示、查询以及它们之间关系进行建模的抽象描述方法，它由三元组 $F[D, \mathrm{Q}, \mathrm{R}(\mathrm{q}_i, \mathrm{d}_j)]$ 表示。其中，D 是文档集中的一组文档逻辑视图(或称为文档的表示)；Q 是一组用户信息需求的逻辑视图(表示)，这种视图(表示)被称为查询；$\mathrm{R}(\mathrm{Q}_\mathrm{I}, \mathrm{D}_\mathrm{J})$是一个排序函数，该函数输出一个与查询 $\mathrm{q}_i \in \mathrm{Q}$，文档表示 $\mathrm{d}_j \in \mathrm{D}$ 有关的实数，用来表示查询和结果文档之间的相似度。

根据查找相关信息的实现方式不同，经典的信息检索模型有布尔模型(Boolean Model)、向量空间模型(Vector Space Model)和概率模型(Probabilistic Model)。布尔模型是许多商业信息检索系统的理论基础。在布尔模型中，文档和查询都被表示为索引项的集合，因此该模型是集合论模型；在向量空间模型中，把文档和查询表示成一个 n 维空间中的向量，用距离作为相似度的度量，该模型是代数模型；在概率模型中，把检索看作是文档和查询之间匹配成功的概率估计问题，用于构建文档和查询的机制是基于概率论的，该模型是概率模型。

5.2.1　集合模型

布尔模型是最典型的一种集合模型，是一种简单的匹配模型，如果用户提交的查询词条在文档中出现，就赋予 True 值，反之为 False；用 AND、OR、NOT 等逻辑运算还可以将 n 个查询词条连接成一个逻辑表达式。在布尔模型中，如果查询表达式为真，RSV(Retrial Status Value)值就为 1，否则赋予 0，除了 0、1 这两个值以外，没有第 3 个值。因此利用布尔模型无法在匹配集中进行相关性的排序。布尔模型易于实现，检索速度快，几乎所有的商业站点都支持该模型。但是布尔模型过于严格，往往由于一个词条不满足条件而导致整个查询表达式无效，漏检比较严重，同时也无法区分词条在文档中所占的权重，因此布尔模型是一种简单但是不够理想的快速检索模型。

5.2.2　代数模型

向量模型把文档看成一组独立的 n 维词条向量($v_1, v_2, \cdots, v_n$)，对每一个词条分量都赋予一个权值 w_i，文档和用户的信息查询可以转化为向量空间的向量匹配问题[1-2]。假设文档向量为($w_1, w_2, \cdots, w_n$)，用户查询为($q_1, q_2, \cdots, q_n$)，用两个向量的夹角余弦来表示文档的相似度，很明显该角度与文档的相似度成反比。文档特征向量($w_1, w_2, \cdots, w_n$)可以根据词条的频度来选取。一般来说，词条的重要程度和词条在文档中的出现频率成正比，和文档集合中出现该词条的文档数量成反比。只要满足上述条件，就可以构造一些函数来具体计算。

P 范式模型可以看成一种扩展的布尔模型。在 P 范式模型中，文档同样可以表示成一组词条向量($w_1, w_2, \cdots, w_n$)，其中 w_i 表示第 i 个词条在文档中的权重，它的取值在[0,1]之间，当 $P=\infty$ 时，该模型就退化为布尔模型。在实际应用中 P 的取值一般在 2～5。

5.2.3　概率模型

概率、模型比布尔模型、向量模型改进的是考虑了词条文档之间的统计

概率[1]。假设 N 表示文档集合中的文档总数，R 表示文档集合中与用户查询相关的文档数，N_i 表示包含查询词条 T_i 的文档数，R_t 表示相关文档 R 中包含词条 T_t 的文档数。文档相似度的计算都和这些变量相关，词条、文档之间的内在关联在该模型中得以体现。

概率模型试图在一个概率框架中处理信息检索问题，其基本思想是：给定一个用户的查询，则有一个包含相关文档且不包含不相关文档的集合。设想这个文档集合是一个理想的结果集，给出这个理想结果集的描述，并用于检索。

基本假设：给定一个查询 q 和文档集中一个文档 dj，概率模型试图找出用户对其感兴趣的概率，模型假设这个概率只是依赖于查询和文档的表示，进而模型假设文档集中存在一个子集，它使得总体相关概率，在集合中的文档被认为是与查询相关的，不在集合中的则被认为是不相关的。

其主要优点是：理论上，文档按照其与目标集合的相关概率的降序排列。

主要缺点是：需要最初将文档分为相关和不相关的集合；所有权重都是二值的，模型中仍然假设索引项之间是相互独立的。

5.2.4　概念模型

概念模型是一种全新的检索模型，该模型的基本原理和上述的几种模型有本质的区别[3-5]。该模型不是以单词或词组为中心组织检索数据库，而是以概念来取代它们，用树状或网状结构来表示概念的组织和分类。使用概念模型检索，就不再局限于词条本身，当用户输入一个查询条件时，不仅要找出与查询表达式匹配的结果，而且搜索引擎根据该词语概念与其他词语概念的内在关联，也要找出包含与查询表达式概念相同或相近的词语的文档。例如，用户查找 color change，搜索引擎可以找出 become black 等具有相近含义的词语。概念模型不是简单的短语匹配，这种短语匹配往往得到成千上万毫无意义的结果，但是它能够根据用户查询词条的内在含义进行相近语义短语的查找，这个特点是其他的模型所没有的。

5.3　N 层向量空间模型

向量空间模型是目前信息检索最常用的数学模型[6]。在 Web 信息检索方面，向量空间模型比布尔模型等传统模型更为适合。

基于向量空间模型的信息检索一般过程如下：

(1) 将各个文档和查询都表示为向量；

(2) 计算查询与各个文档之间的相似度；

(3) 按照查询与各个文档之间的相似度对相关的文档进行排序；

(4) 将排序后的文档以线性列表的形式返回给用户。

5.3.1　向量空间

在向量空间模型中，各个文档(document)和查询(query)的内容都表示为向量。设共有 n 个文档，对所有文档进行词语切分、过滤和转换之后，合适的 m 个词语(term)被提取出来，构造 $m\times n$ 的词语——文档关系矩阵(term-document matrix)A，矩阵中的元素 $A(i,j)$表示第 $i(1\leqslant i\leqslant m)$个词语 T_i 在第 $j(1\leqslant j\leqslant n)$个文档 D_j 中的权重(weight)，这样，每个词语就可以用 A 中对应的行向量来表示，每个文档就可以用 A 中对应的列向量来表示。所有文档都是位于 m 维空间中的向量，查询也可以与文档一样表示为 m 维空间中的向量。

5.3.2　权重

词语 T_i 在文档 D_j 中的权重 $A(i,j)=L(i,j)\times G(i)$，其中 $L(i,j)$是局部权重(Local Weight)，体现了词语 T_i 对文档 D_j 的重要程度，$G(i)$是全局权重(Global Weight)，表示了一个词语区分其他文档的能力。

常见的局部权重有[6-7]：

- TF：$L(i,j)=tf_{ij}$
- Log TF：$L(i,j)=\log_2(tf_{ij}+1)$

常见的全部权重有：

- GF * IDF：$G(i) = gf_i / df_i$
- IDF：$G(i) = \log 2(n_d / df_i) + 1$
- Entropy：$G(i) = 1 - \sum_j \frac{p_{ij} \log_2(p_{ij})}{\log_2(n_d)}$，$p_{ij} = tf_i / gf_i$

其中，gf_i 表示第 i 个词语在所有文档中的出现频率，df_i 表示含有第 i 个词语的文档的数目，n_d 表示所有文档的数目。

在向量空间模型中最常用的权重计算方法是 TF * IDF，因为它容易实现而且速度较快。

表示索引项 $\boldsymbol{T_k}$ 对文档 $\boldsymbol{D_i}$ 的重要程度，记为 $\boldsymbol{w_{ik}}$。

$$w_{ik} = L(i,k) \times G(i) \tag{5-1}$$

其中 $\boldsymbol{L(i,k)}$ 代表索引项 T_k 在文档 D_i 中的局部权重，$G(i)$ 为索引项 $\boldsymbol{T_k}$ 的全局权重。计算方法一般采用通用的 if-idf 公式：

$$w_{ik} = \frac{tf_{ik} \times \log(N/df_{ik} + 0.01)}{\sqrt{\sum_{T_k \in D_i} [f_{ik} \times \log(N/df_k + 0.01)]^2}} \tag{5-2}$$

其中，tf_{ik} 表示索引项 $\boldsymbol{T_k}$ 在文档 D_i 中出现的次数（即索引项频率），tf_{ik} 越高，意味着索引项 $\boldsymbol{T_k}$ 对于文档 D_i 越重要；df_k 表示含有索引项 $\boldsymbol{T_k}$ 的文档数量（即索引项的文档频率），df_k 越高，意味着索引项 $\boldsymbol{T_k}$ 在衡量文档之间相似性方面的作用越低；$\boldsymbol{N} = |\boldsymbol{D}|$，即全部文档的数量，分母为归一化因子；$idf_k = \log(N/df_k)$ 为逆向文档频率，idf_k 越高，意味着索引项 $\boldsymbol{T_k}$ 对于文档的区别作用越大。如果一个索引项 $\boldsymbol{T_k}$ 仅出现在一个文档中，则 $idf_k = \log(N)$。

如果一个索引项 $\boldsymbol{T_k}$ 出在所有的文档中，则 $idf_k = \log 1 = 0$。

5.3.3 文档和检索项之间的相关性

文档与查询之间的相关性可以用文档向量与查询向量之间的相似度（similarity）来度量。在向量空间模型中最常用的相似度是余弦系数（cosine coefficient），也就是文档向量与查询向量之间夹角的余弦。根据线性代数知

识,向量之间夹角的余弦实际上就是将两个向量规范化(normalize)后的内积(inner product)。

5.3.4　*N* 层向量空间模型(N-Vector Space Model)

在经典的向量空间检索模型中,文档和查询都是用其所包含的特征项组成的向量来表示的,并且用文档与查询的向量之间夹角的余弦作为相似性的度量,夹角越小,相似度越大。针对特定的查询向量,比较它与所有文档向量的相似度,并依相似度将文档降序排序,提交检索结果。由于文档集合中特征项的数量远远大于每一篇文档或查询中特征项的个数,因此文档和查询的向量表示形式中的大部分项都为零。这些零项将会在计算特征项的权重和相似度时带来很大的时间和空间复杂度,导致数据稀疏现象。另外在特征项抽取以及查询匹配过程中,同一个特征项出现在文档的不同区域时,它所表达文档内容的能力是不同的,而且在文档同一区域,不同的特征项所表达文档内容的能力也是有差别的。使用传统的向量空间模型则会认为这些特征项所表达文档的能力完全相同,不能加以区分。为提高向量空间模型的检索效率,陈治平[8]等提出了 *N* 层向量空间模型的信息检索算法。将一篇文档从组织结构上划分为 *N* 层,基于每层的文本内容建立相应的特征项向量和权值。其中特征项抽取和权重计算等同传统向量空间模型相同,这样对于文档进行 *N* 层划分得到的向量空间模型就成为 *N* 层向量空间模型。

5.4　N-VSM 的文档相似度计算

设文档集合中共有 n 个不同的索引项 $T_1,T_2,\cdots,T_n$,根据权重计算公式计算文档 $\boldsymbol{D}_i(\mathbf{1},\mathbf{2},\cdots,\boldsymbol{m})$的索引项权重 w_{ik},如果把文档 $D_1,D_2,\cdots,D_n$ 看成一个 n 维坐标系,w_{ik} 为坐标系的值,则 $\boldsymbol{d}_i=(w_{i1}^{(d)},w_{i2}^{(d)},\cdots,w_{in}^{(d)})^T$ 成为 n 维空间中的一个向量,即文档 $\boldsymbol{D}_i$ 的向量表示如表 5-1 所示。

表 5-1 文档的向量空间模型

文档集合	文档内容特征(词)			
	T_1,	T_2,	…	T_m,
D_1	w_{11}	w_{12}	…	w_{1m}
D_2	w_{21}	w_{22}	…	w_{2m}
…	…	…	…	…
D_k	w_{k1}	w_{k2}	…	w_{km}
…	…	…	…	…
D_n	w_{n1}	w_{n1}	…	w_{nm}

计算相似度的方法有许多种,通常用两个向量的夹角来表示文献的相似度(Jaccard 相似度函数),其中 Jaccard 相似函数为:

$$S(q_j,d_i)=\cos\theta=\frac{\sum_{k=1}^{n}w_{jk}^{(q)}w_{jk}^{(d)}}{\sqrt{\sum_{k=1}^{n}(w_{jk}^{(q)})^2+\sum_{k=1}^{n}(w_{jk}^{(d)})^2}} \tag{5-3}$$

5.4.1 常见的特征权重算法

特征项在不同 Web 文本中出现的频率满足一定的统计规律,因此常常通过特征项的频率特性计算其权重。关键词的权重的计算方法有很多:布尔函数、开根号函数、TF 函数、IDF 函数、TFIDF 公式法。

1. 特征项频率(Term Frequency,TF)

TF(Term Frequency)是频率特性,它指的是当前研究的频率的文档。单词和短语,也可以是语义概念字典语义合并后词语权重计算方法或概念语义单位。不同类型的文件的某些特征频率是非常不同的项目,所以频率信息是一种重要的参考,一般在文本分类技术价值更大的特色项目在这个文件具有较高的重量。在原文本自动分类、文献载体是用来构建技术。其计算公式为:

$$w_i=TF(t_i)=\text{特征项 } t_i \text{ 在类 } c \text{ 文本中出现的次数}$$

考虑到获取信息的特点,使每一体重之间有较大差别重量法的比率,具有明显的进步,但只考虑频率信息会导致过分依赖高频单词和可以放弃一些很多信息的低频的话,在实际应用中,由于各种不文本的长度是很难同意,各种各样的文字,文字包含数字可能差别会很大、频率会造成直接的影响,我们通常将词频正常化。

2. 反文档频率(Inverse Document Frequency,IDF)

简单地说,使用技术来记录有用的程度,一方面,文件会大量出现在一些分类中没有贡献的功能(如:感叹、介词、连词,等等),这些话出现频率一般比较大,根据 TF 重量计算方法可以选择这些没有很多的信息功能分类特征;另一方面,特征词的好坏是否能代表类和文档的属性。TF 值高的特征词,如果在所有的文档中 TF 值都高,那就很难说这样的特征词到底代表哪个类或者文档的属性。

IDF 出现网络文本数为参数的特色项目建设的权重。IDF 重量法的出发点是一个网络文献频率较高,它包含类别信息就越低。

IDF 的计算方法如公式(5-4)所示:

$$\boldsymbol{IDF}(t_i) = \log\left(\frac{N}{n_i} + L\right) \tag{5-4}$$

其中,L 的取值通过实验来确定(通常取 0.01)。表示为文件收集的总证明文件号码,项目的文件数字就会呈现出来的特点。IDF 的核心理念,在大多数文件算法不仅特色项目中都出现在一小部分文件出现在重要的特性。IDF 算法可以在大多数文件会少了一些高频特性均出现的重要程度,同时提高一些在小文件出现在低频率特性的重要程度。

5.4.2　TD-IDF 加权公式

1. TF-IDF 函数

针对这些权重计算方法中所存在的局限性,人们往往将反文档频率 IDF 和 TF 结合使用,也就是 TF-IDF 权重,TF-IDF 的归一化计算方法如公式(5-5)

所示：

$$w_i = \frac{TF(t_i) \times IDF(t_i)}{\sqrt{\sum_{i=1}^{n}(TF(t_i) \times IDF(t_i))^2}} \tag{5-5}$$

其中 w_i 为第 i 个特征项在文本中的权重，$TF(t_i)$是特征项 t_i 在文本 d_j 中出现的频数，n 表示特征向量的维数。

由公式(5-5)可以看出 TF-IDF 权重计算方法中主要考虑了三个因素：

(1) 词频 TF。通过 TF 的定义可以看出，一个特征项如果在某一类别里出现的频率很高，那么这个特征项可能对于该类别很有代表性，TF 较大的特征项就在该类文档中赋予一个较高的权重，所以 TF-IDF 的权重计算方法考虑了 TF 值。

(2) 反文档频率 IDF。IDF 考虑的是如果一个特征项在较少文档里面出现，那么它可能更能代表某一类别的信息，所以 TF-IDF 的计算还需要计算 IDF 值。

(3) 对于每个部件标准化归一化的因素。由于各种文本的长度是很难统一，各种各样的文字，文字包含一些颜色，它将有很大的差异，所以获得直接影响词频为正常化。

2. 标准 TF-IDF 特征权重计算分析

TF-IDF 方法中 IDF 值的计算将训练文档集作为一个整体来考虑，并没有考虑到特征项在类间的分布信息。比如说，如果某一类 c_i 中包含词条 T_k 的文档数为 a，而其他类包含互的文档总数为 b，那么，所有包含 T_k 的文档数为 $n=a+b$，显然，a 越大，n 也越大，根据公式(5-4)可以得到，IDF 的值越小，则根据 TF-IDF 的计算方法，权重值也会受到影响。但是实际上，a 大的话，表示特征项 T_k 更频繁地在类 c_i 出现，很明显，T_k 能更好地代表 c_i 类的文本特征，所以在计算文本属于 c_i 类的后验概率的时候，应该给词条赋予比较高的权重，从而体现出 T_k 对于 c_i 类的重要性。

考虑改进基于加权的朴素贝叶斯分类算法中的权重计算方法，特征权重的调整是为了强调辨别能力强的特征，抑制没有或辨别能力低的特征。

如果采用传统的 TF-IDF 权重计算方法，那么对于某一篇测试文本，特征项的 TF 值可根据测试文本获得，这样该特征项的权重就由 IDF 值来进行调整了，而 IDF 不能很好地反映特征的重要性，因为它的值是基于将训练集的文档看成一个整体来考虑的，所以不能表示特征项与类别之间的关联性。

因此不能表示出特征项和类别之间的关联性。然而，所研究的基于加权的朴素贝叶斯分类算法里权重的作用主要是为了体现不同特征项在分类时所起作用的不一样，那么用 TF-IDF 来计算权重显然不是太合理。

5.4.3　基于贝叶斯理论的加权计算

通过上面的例子说明了传统 TF-IDF 特征权重计算方法所存在的一些弊端，主要表现在该方法中 IDF 值的计算是将类别中的文档作为一个整体来进行考虑的，而没有考虑特征项在类别中的分布情况这一重要信息。而向量空间模型可以很好地表征特征项在 Web 文档不同位置的重要程度，为了弥补传统 TF-IDF 特征权重计算方法的缺陷，将函数 RTC[9] 引入到向量空间模型来表示特征项与类之间的关联性，在权重的计算公式里加入特征项在类别中的 RTC 值。特征项 w 的期望交叉熵定义公式(5-6)为：

$$f_{Cr}(w) = p(w)\sum_{i} p(c_i \mid w)\log\frac{p(c_i \mid w)}{p(c_i)} \tag{5-6}$$

其中，$p(c_i|w)$表示文本中出现特征项 w 时，文本属于类别 c_i 的概率，$p(c_i)$是类别出现的概率。如果特征项和类别强相关，也就是 $p(c_i|w)$大，且相应的类别出现概率又小的话，则说明词条对分类的影响大，相应的函数值就大，就可能被选中做出特征项。交叉熵反映了文本类别的概率分布和在出现了某个特征词的条件下文本类别的概率分布之间的距离，特征项 w 的交叉熵越大，对文本类别分布的影响也越大。我们要计算的权重吆表示的是特征项在类别中的权重，所以这里采用函数 RTC 来表示期望交叉熵计算公式中 $p(c_i|w)\log\frac{p(c_i|w)}{p(c_i)}$的计算部分，用来衡量特征项和类别之间的关联

性。RTC 的计算如公式(5-7)所示：

$$\boldsymbol{RTC}(c_i, w) = p(c_i \mid w)\log \frac{p(c_i \mid w)}{p(c_i)} \tag{5-7}$$

该公式表示是特征项 w 在 c_i 类下的 RTC 值。当概率 $p(c_i|w)$ 较大，而 $p(c_i)$ 又较小的时候，算出的 RTC 值比较大，也就是表示特征项和类别的关联性比较强。

下面我们将特征项和类别的 RTC 值考虑到特征权重的计算公式当中去，提出一种新的权重计算方法 TF-IDF-RTC，该方法考虑了特征项在不同类别中的 RTC 值，计算方法如公式(5-8)所示：

$$w_{ik} = \frac{TF(t_i) \times IDF(t_i) \times RTC(c_k, t_i)}{\sqrt{\sum_{i=1}^{n}(TF(t_i) \times IDF(t_i) \times RTC(c_k, t_i))^2}} \tag{5-8}$$

其中 w_{ik} 表示的是特征项 t_k 在类别 c_k 下的权重。

5.4.4 基于 *N* 层向量空间模型的文档检索

由于 Web 文献的特殊格式[10-11]，要求一篇文档最少是由文档标题、摘要、关键词和文档正文四部分组成，而这四部分的内容对于这篇文档的表达能力是不同的，文献[12]通过对随机采集的 300 篇中文经济类网页进行人工自由标引、人工打分、词频统计，并进行统计数据的分析，得出网页内容主题与网页题名、文章标题等 12 个标引源的关系，分析中文网页的不同部位的主题表达能力，并为之设计加权标引时的适当权值，以便为自动标引及人工智能搜索引擎的研制提供数据，为考虑索引项出现在不同位置的权重设置提供了理论基础。

在综合考虑标题、摘要、关键词和正文四个位置，构建四层向量空间模型，并考虑了某个特征项同时出现在标题和关键词的现象。论文是科学研究中创造性思想的载体，它的首要任务在于传递科研信息，同时也具有文化储存和文化积累的意义。无论是从传递信息角度，还是储存信息角度考虑，主题词或关键词的标引都将给文献的储存和检索带来极大的方便。

为了确定索引项在不同位置的权重，假设在一个 Web 文档集中有 n 个文档 $D1,D2,D3,\cdots,Dn$，这 n 个文档中都包含索引项 T，并且 T 在这 n 个文档中出现次数都相同。如果在文档 Di 中，T 是被包含在标题中，在文档 Dj 中，T 是被包含在摘要中，在文档 Dk 中，T 同时被包含在标题和关键词中，在 Dm 中，T 是被包含在正文中，运用传统的信息搜索引擎则会认为索引项 T 表达这几个文档的能力完全相同。根据以上分析结果，出现在标题中的索引项要比出现在摘要中的索引项更能确切代表文档的内容，同样出现在摘要中的索引项也要比出现在正文中的索引项更能代表文档的内容，而同时出现在标题和关键词中更能确切的代表文档的内容。

为了更好地提高用户的检索要求，采取如下步骤改进检索模型：

(1) 从文档中抽取关键词，根据公式(5-8)计算权重；

(2) 将用户检索项和待检索文档表示成向量空间中的向量；

(3) 根据检索项在文档不同位置(标题、摘要、关键词和正文)构建四层向量空间模型，同时考虑在关键词和标题同时出现的情况；

(4) 对检索项在不同位置出现的情况赋予不同的权重，利用夹角余弦来计算相似度，得到检索兴趣度最高的文本。

处理过程如图 5-2 所示。

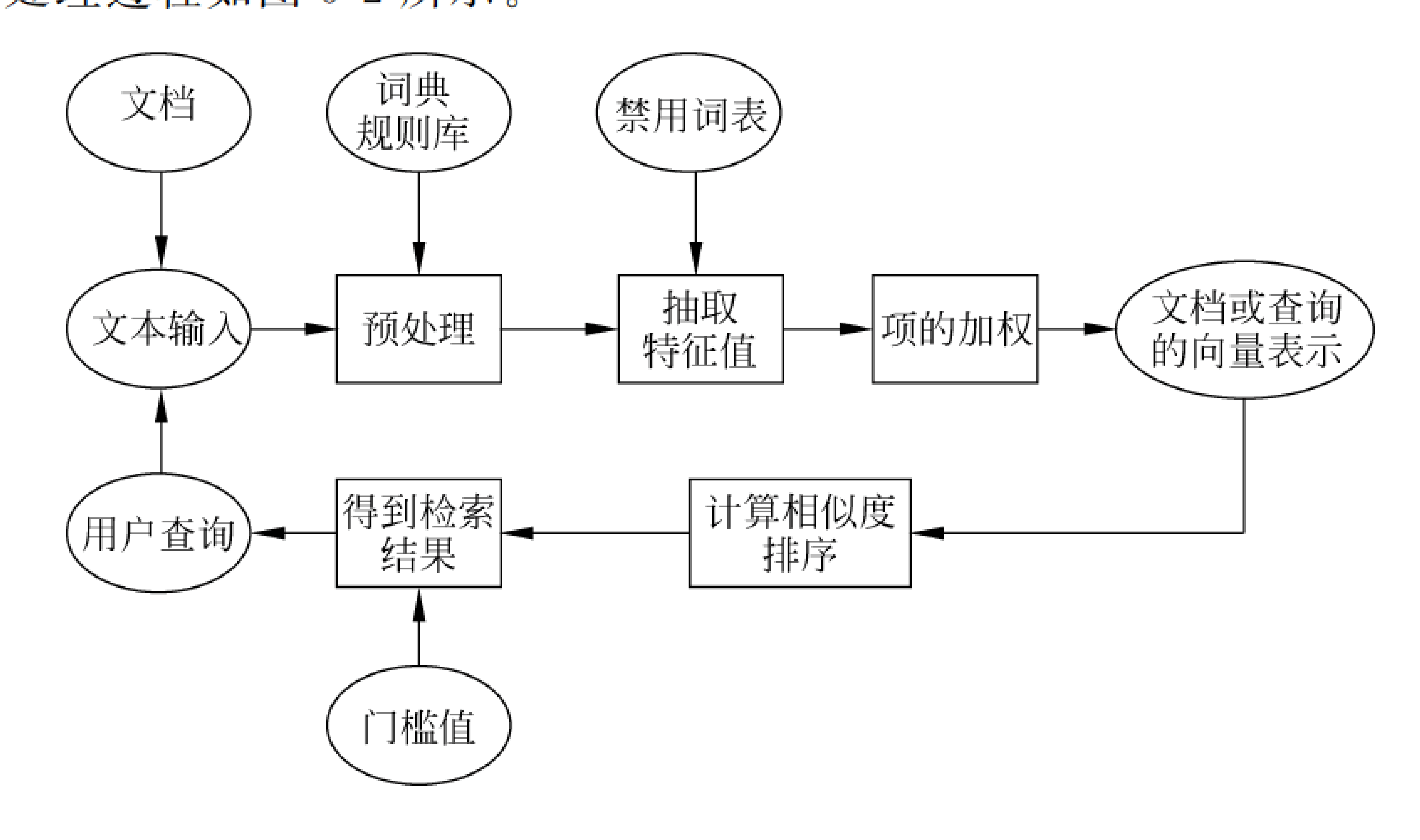


图 5-2　N 层向量空间系统处理图

文档的 N 层向量空间表示，如表 5-2 所示。

表 5-2　N 层向量空间模型

文档集合	文档内容特征(词)			
	T_1	T_2	…	T_m
D_1	$w_{11} * \eta_{1j}$	$w_{12} * \eta_{2j}$	…	$w_{1m} * \eta_{mj}$
D_2	$w_{21} * \eta_{1j}$	$w_{22} * \eta_{2j}$	…	$w_{2m} * \eta_{mj}$
…	…	…	…	…
D_k	$w_{k1} * \eta_{1j}$	$w_{k2} * \eta_{2j}$	…	$w_{km} * \eta_{mj}$
…	…	…	…	…
D_n	$w_{n1} * \eta_{1j}$	$w_{n1} * \eta_{2j}$	…	$w_{nm} * \eta_{mj}$

其中，η_j 为索引项 $\boldsymbol{T_k}$ 在文档 $\boldsymbol{D_i}$ 出现在第 j 层在权重，$j=1,2,3,4$。

5.5　检索结果排序算法

信息检索的核心问题之一是在系统检索出的结果集中运用排序算法对检索结果按照一定的相关性进行排序。排序函数是以某种准则计算文档信息与用户检索信息的相关性判断，根据相关度进行降序排列。排序技术是信息检索系统进行结果处理的核心技术，排序策略直接影响到检索系统的查准率和查全率，排序算法的优劣直接影响系统的效率。典型的检索系统排序技术主要有词频统计和词位置加权排序算法、基于用户反馈的 Direct Hit 算法、PageRank 链接分析算法和 Hits 排序算法四种，早期搜索引擎如 Infoseek、Lycos 等采用的是词频统计和词位置加权排序技术，现在流行的两大搜索引擎 Google、百度采用的是超链分析排序技术。本节主要分析讨论基于链接的排序算法，并对排序函数做了改进研究。

5.5.1　超链分析排序技术

人们在传统信息检索排序技术的基础上，结合网络信息检索自身特性，提出了许多 Web 页面检索排序算法，主要分为三类：基于网页内容的排序

算法、基于网页链接分析的排序和基于检索用户的排序算法，在实际应用中往往会综合应用这些算法。

最著名的基于链接分析的算法是由 Page 提出的 PageRank 算法和 Kleinberg 提出的 HITS 算法，且 PageRank 在 Google 中的应用获得了巨大的商业成功。由于 Web 文档涉及参考文献的引用问题，所以将其看作是一种基于链接的信息资源，所以对现有的 PageRank 算法进行了分析和综述，并将其引入 Web 文档检索的向量空间模型[13]。

1. PageRank 算法

基本思想：如果网页 T 存在一个指向网页 A 的连接，则表明 T 的所有者认为 A 比较重要，从而把 T 的一部分重要性得分赋予 A。这个重要性得分值为：PR(T)/C(T)，其中 PR(T)为 T 的 PageRank 值，C(T)为 T 的出链数，则 A 的 PageRank 值为一系列类似于 T 的页面重要性得分值的累加。

优点：是一个与查询无关的静态算法，所有网页的 PageRank 值通过离线计算获得；有效减少在线查询时的计算量，极大降低了查询响应时间。

不足：人们的查询具有主题特征，PageRank 忽略了主题相关性，导致结果的相关性和主题性降低；另外，PageRank 有很严重的对新网页的歧视。

2. Topic-Sensitive PageRank(主题敏感的 PageRank)

基本思想：针对 PageRank 对主题的忽略而提出。核心思想：通过离线计算出一个 PageRank 向量集合，该集合中的每一个向量与某一主题相关，即计算某个页面关于不同主题的得分。主要分为两个阶段：主题相关的 PageRank 向量集合的计算和在线查询时主题的确定。

优点：根据用户的查询请求和相关上下文判断用户查询相关的主题(用户的兴趣)返回查询结果准确性高。

不足：没有利用主题的相关性来提高链接得分的准确性。

3. Hilltop

基本思想：与 PageRank 的不同之处是仅考虑专家页面的链接。主要包括两个步骤：专家页面搜索和目标页面排序。

优点：相关性强，结果准确。

不足：专家页面的搜索和确定对算法起关键作用，专家页面的质量决定了算法的准确性，而专家页面的质量和公平性难以保证；忽略了大量非专家页面的影响，不能反应整个 Internet 的民意；当没有足够的专家页面存在时，返回空，所以 Hilltop 适合对于查询排序进行求精。

影响 google PageRank 的因素：

(1) 与 pr 高的网站做链接；

(2) 内容质量高的网站链接；

(3) 加入搜索引擎分类目录；

(4) 加入免费开源目录；

(5) 你的链接出现在流量大、知名度高、频繁更新的重要网站上；

(6) Google 对 PDF 格式的文件比较看重；

(7) 安装 Google 工具条；

(8) 域名和 tilte 标题出现关键词与 meta 标签等；

(9) 反向连接数量和反向连接的等级；

(10) Google 抓取您网站的页面数量；

(11) 导出链接数量。

5.5.2 Page Rank 算法分析

简短的等级的算法是基于网络图模型，整个网页结构逻辑可作为一种巨大的有向图(V,E)，V 是节点集，每一页都是这个数字在一个节点，页面可以看作是网络信息的载体。E 是一家集向边，如果你的页面 v1 包含一个指针指向页 v2 超链接，即存在一条有向边 Hy-perlink(v1 与 v2)。用超链接观

众之间,可以浏览页面的页和超文本链接构成巨大的有向图。

PageRank 算法从文学最初思想理论,即参考文献的重要性可以利用文献的数量和质量计量,引证了说明文献参考价值,更有价值的文献引用说明文学的重要性越高。简短的排名算法可以被描述为:

$$PR(u) = c \sum_{v \in B(u)} \frac{PR(v)}{N(v)} + cE(u) \tag{5-9}$$

式中 $PR(u)$表示页面 u 的 PageRank 值(重要度)。v 为指向页面 u 的页面,$B(u)$为指向 u 的页面集合,$N(v)$为页面 v 的导出超链接数,c 是取(0,1)之间的规范化因子,$E(u)$是衰减因子。由此可以得出这样的结论,页面的重要度来自指向它的页面的重要度,同时,页面的重要度被平均分配给页面的导出链接。

PageRank 算法的优点是避免计算纯文本匹配的重要性的页面,利用网络模型超链接页面的数量和质量评估页面的权力,为客观地评价网络资源的有效手段,也可以看到的 PageRank 算法是一种和查询关键词无关的静态的网页质量的计算方法。

利用向量空间模型(Vector Space Model)来计算检索关键字和网络文档关联。在传统信息检索系统、向量空间模型计算相似度计算模型的文件之间,在这个模型的文档,向量来的术语文件由合金的成分、表达权值向量的词元权重,通常是词出现的频率。

5.5.3　Page Rank 算法的改进

1. 理论分析

采用 N 层向量空间模型表示的文档均为矩阵(如表 5-2 所示),为计算方便,采取马尔科夫链的转移概率来简化矩阵的运算。对于由检索项构成的矩阵,用检索项出现的频率来代替随机概率,以改进现在的基于文本链接的 Page Rank 算法。

具体过程如下,由于改进的 N 层向量空间模型,不仅考虑了检索项出现在不同位置的情况,还考虑在不同位置同时出现的情况,将同时出现的情况

看做是在某一位置出现的条件概率，这样对于上一章的权重计算就可以看做是一个马尔科夫链，即得到权重矩阵以后进行转置，把各个列向量（同一个检索项出现在不同的文档）处以出现该检索项的文档总数，这样得到检索项推移概率矩阵，各个行向量表示同一个文档出现不同检索项的概率（频率）。转置在于，对于同一个检索项，排序算法不考虑两个文档之间的引用关系。

2. 算法设计

针对改进后的 N 层向量空间模型，对相似度相同的结果进行第二次排序。由于 N 层向量空间的处理对象是矩阵，采用转移概率计算，以简化其计算过程。

算法描述：

（1）通过权重函数计算检索项的权重矩阵；

（2）计算转移概率矩阵；

（3）求得各个检索项的转移概率；

（4）进行迭代，直到马尔科夫链收敛。

算法简例：如果我们通过改进后的权重计算公式得到如下权重矩阵：

$$M=\begin{pmatrix} 0 & 1 & \frac{1}{2} & 0 & \frac{1}{4} & \frac{1}{2} & 0 \\ \frac{1}{5} & 0 & \frac{1}{2} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{5} & 0 & 0 & \frac{1}{3} & \frac{1}{4} & 0 & 0 \\ \frac{1}{5} & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 \\ \frac{1}{5} & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{2} & 1 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 \\ \frac{1}{5} & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

而通过该算法计算后得到的转移矩阵为：

$$M' = \begin{pmatrix} 0 & 1 & 0.6735 & 0 & 0.6222 & 0.7098 & 0 \\ 0.3133 & 0 & 0.3265 & 0.3766 & 0 & 0 & 0 \\ 0.2543 & 0 & 0 & 0.3057 & 0.2448 & 0 & 0 \\ 0.1326 & 0 & 0 & 0 & 0.1276 & 0 & 0 \\ 0.2642 & 0 & 0 & 0.3177 & 0 & 0.2902 & 1 \\ 0 & 0 & 0 & 0 & 0.0054 & 0 & 0 \\ 0.0357 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

该矩阵不论是数学期望还是方差，在其计算上，都比原来的矩阵要简单得多。

5.6　*N* 层向量空间模型权重实验仿真

实验是在基于 Lucene 全文检索引擎[14-15]系统架构下设计的，Web 文献检索的 N 层向量空间模型中，每层的权重对检索结果影响很大，为了更加准确表示，特设置了四个仿真实验，确定四层向量空间模型的权重。

5.6.1　实验数据和实验结果

从维普数据库上下载了有关计算机方面的 Web 文献 1000 篇，从索引库中找出和计算机有关的常用词组 2000 个，以此构建检索数据源的索引库。

将文档分为标题、摘要、关键词和正文四部分，得到四层向量空间模型，并考虑索引项同时出现在标题和关键来设定权重。为方便用户根据自己的需要来控制检索结果和比较不同的权重，采取四个实验来验证改进的模型。同时，为保证查准率进行多次试验证明，门槛值选取在 0.1～0.3，查全率和查准率对比效果比较明显，因此四个试验门槛值均采用 0.2 进行。

1. 实验 1——标题权重

设定 $\eta_2=0.9$（摘要），$\eta_3=0.9$（关键词），$\eta_4=0.8$（正文），$\eta_5=1$（同时出

现在标题和关键词,不用统计词频),而改变标题的权重,实验数据如表 5-3 所示,检索结果如图 5-3 所示。

表 5-3 实验 1 数据表

标题权重	0	0.95	0.9	0.85	0.8
查准率(%)	91.35	93.21	90.46	86.37	820.5

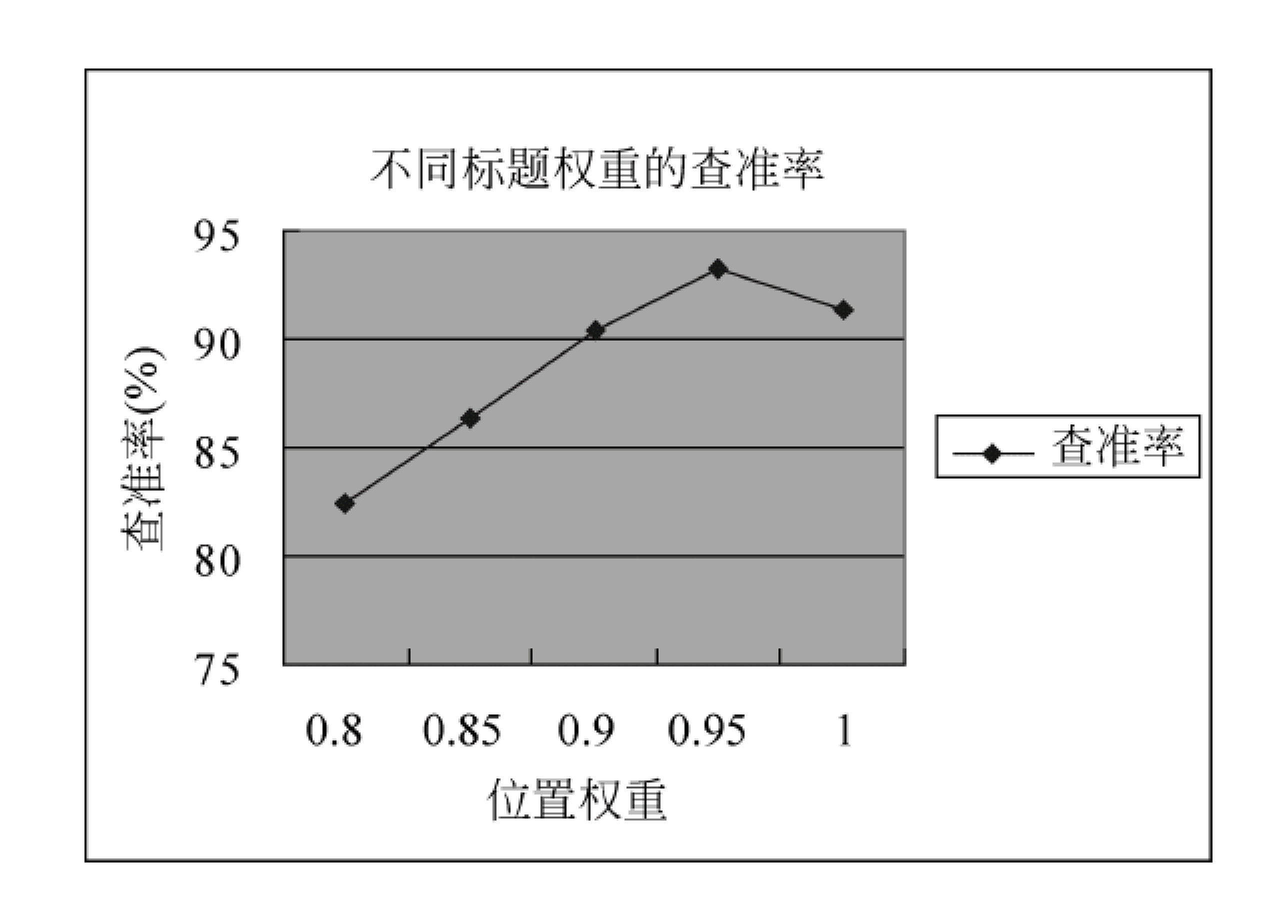


图 5-3 不同标题权重的查准率

2. 实验 2——摘要权重

设定 η_1=0.95(标题),η_3=0.9(关键词),η_4=0.8(正文),η_5=1(同时出现在标题和关键词,统计词频),而改变摘要的权重,实验数据如表 5-4 所示,折线图如图 5-4 所示。

表 5-4 实验 2 数据表

摘要权重	0.9	0.85	0.8	0.75	0.7
查准率(%)	92.81	86.65	83.57	75.68	71.02

3. 实验 3——关键词权重

设定 η_1=0.95(标题),η_2=0.9(摘要),η_4=0.8(正文),η_5=1(同时出现在标题和关键词,不统计词频),而改变关键词的权重,实验数据如表 5-5 所

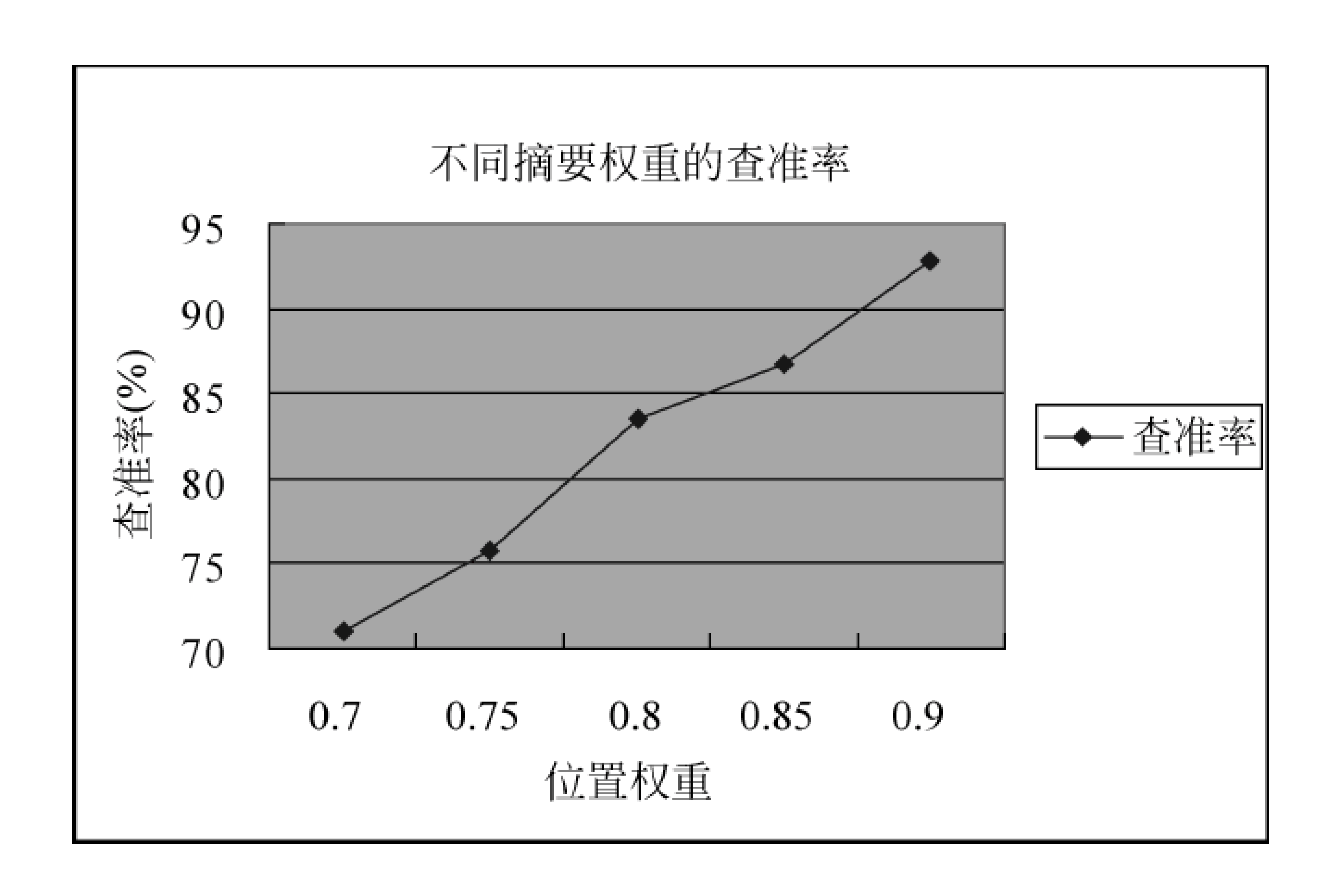


图 5-4　不同摘要权重的查准率

示，折线图如图 5-5 所示。

表 5-5　实验 3 数据表

关键词权重	0.9	0.85	0.8	0.75	0.7
查准率(%)	92.81	93.00	88.94	85.76	80.46

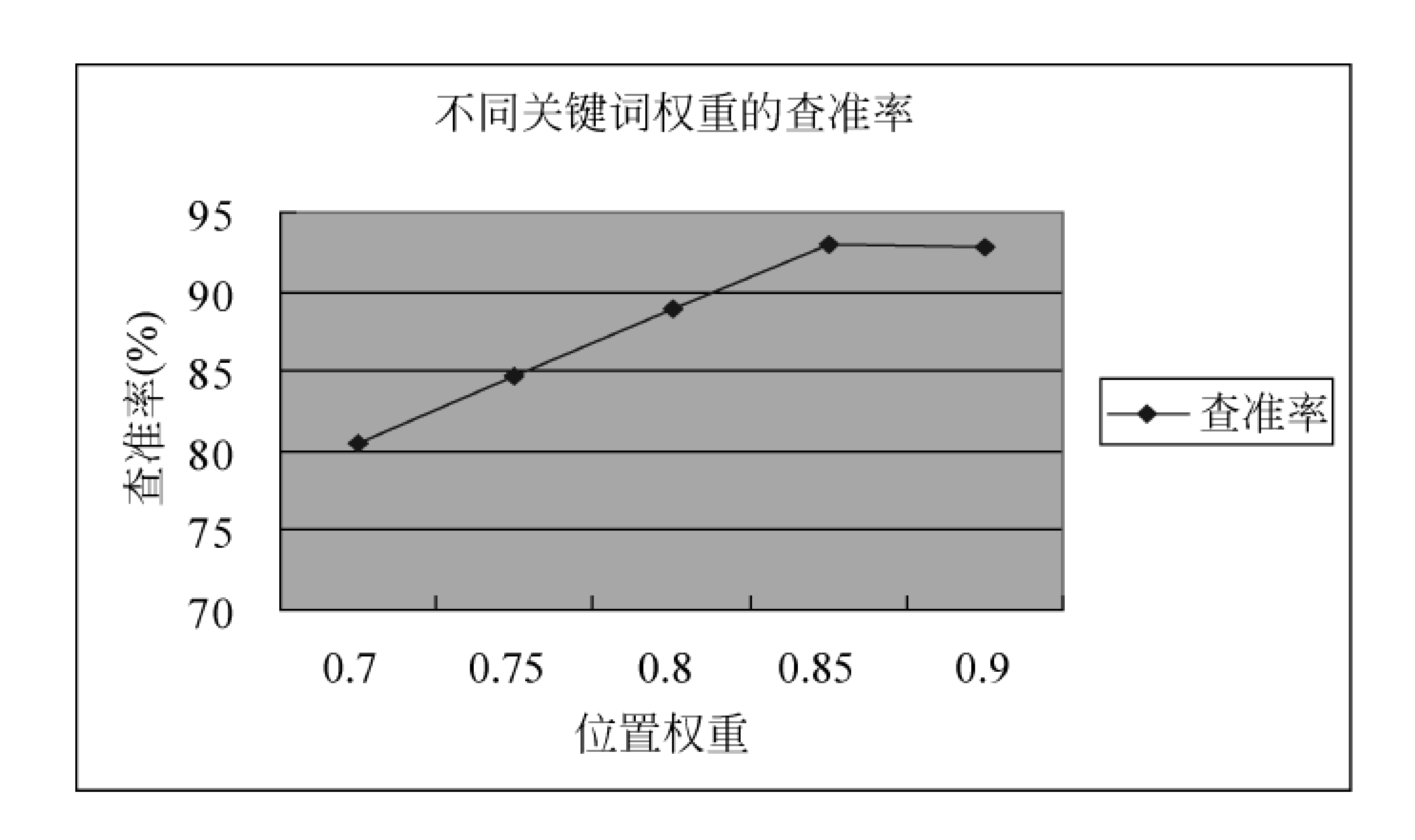


图 5-5　不同关键词权重的查准率

4. 实验 4——标题和关键词权重

设定 $\eta_1=0.95$(标题)，$\eta_2=0.9$(摘要)，$\eta_3=0.85$(关键词)，$\eta_4=0.8$(正文)，改变同时出现在标题和关键词的权重，实验数据如表 5-6 所示，折线图

如图 5-6 所示。

表 5-6 实验 4 数据表

同时出现权重	1	0.95	0.9	0.85
查准率(%)	93.86	91.65	89.94	87.76

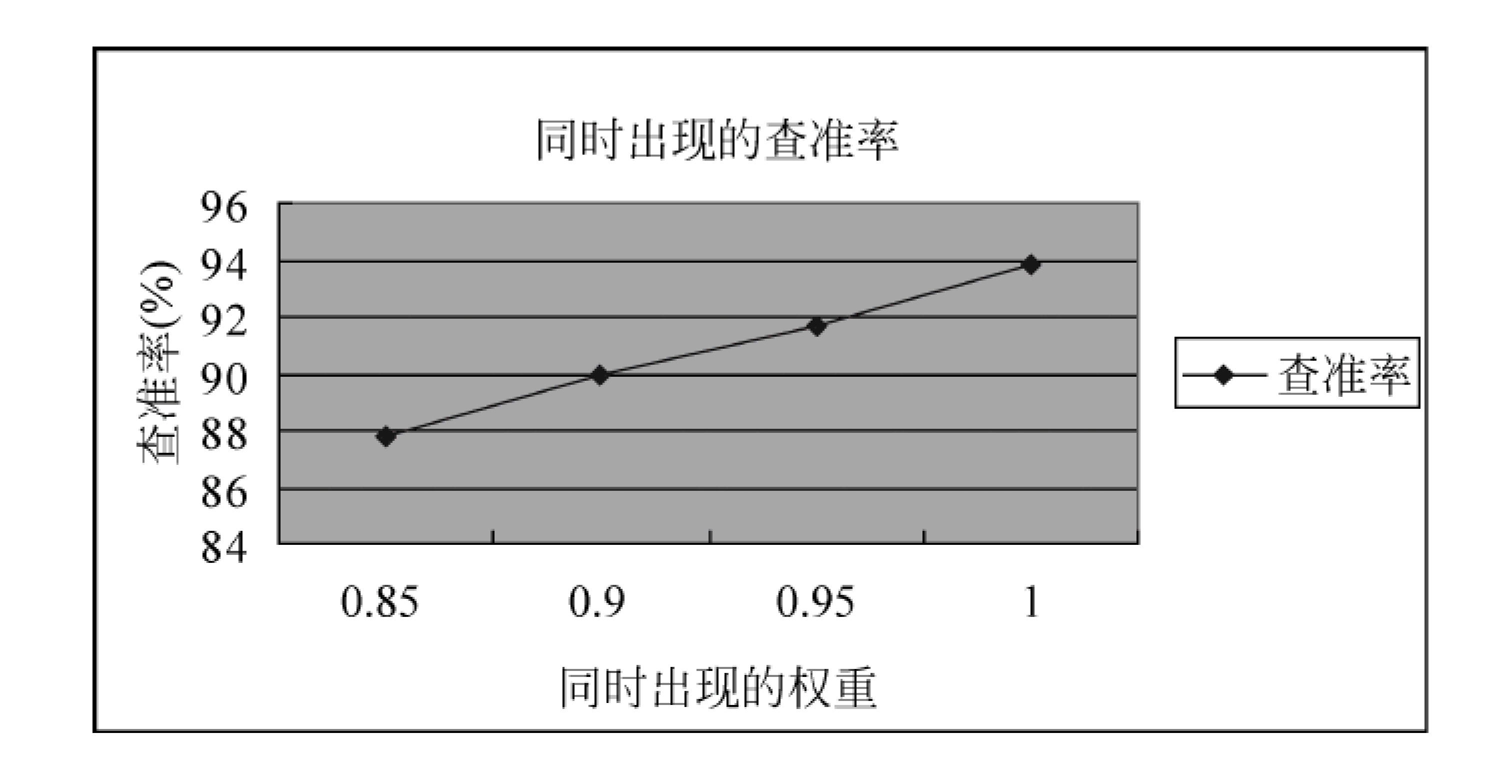


图 5-6 同时出现的查准率

综合实验：根据以上四个实验比较发现，设定 $\eta_1=0.95$（标题），$\eta_2=0.9$（摘要），$\eta_3=0.85$（关键词），$\eta_4=0.8$（正文），$\eta_5=1$（同时出现在标题和关键词，不统计词频），查准率有明显的提高，综合实验数据如表 5-7 所示。

表 5-7 综合实验数据表

门 槛 值	查准率(%)	
	VSM	N-VSM
0.1	78.30	85.63
0.2	86.60	93.36
0.3	93.21	100
0.4	100	100
0.5	100	100
0.6	100	100
0.7	100	100

5.6.2 实验分析

从以上四个试验的结果比较可知，标题的权重控制在 0.85～1 比较合

适,摘要的权重控制在 0.8～0.9 直接比较合适,关键词的权重控制在 0.75～0.9 直接比较合适,而考虑索引项同时出现在摘要和标题的权重在 0.8～1 之间都比较合适。

从以上实验我们可以得到如下结论,由于标题体现出论文的主题观点,所以要选取稍微大的权重,而摘要和关键词体现出论文的整体结构和层次,而且摘要中要考虑索引项的词频,所以可稍微降低一点权重,而考虑索引项同时出现在标题和关键词的情况,将其权重控制在一个合适的范围即可。从实验的数据结果和以上理论分析,本次改进的 N 层向量空间模型,能较好地适应 Web 文献的信息检索,提高了检索的准确性。

参考文献

[1] Gudivada V N, Raghavan V V, Grosky W I. Information Retrival on the World Wide Web. IEEE Internet Computing, 1997: 58～68

[2] Raghavan V, Wong S K M. A Critical Analysis of Vector Space Model for Information Retrieval. J Am Soc Inform Sci, 1986, 37(5): 279～286

[3] Li Z D, Fei X L, Wang H Z. A Concept-Based Information Retrieval Model. Proceedings of the International Symposium on Future Software Technology (ISFST-99). 1999: 296～300

[4] Wiliam A W, Conceptual Indexing: A Better Way to Organize Knowledge. Forthcoming Technical Report. Sun Microsystems Lab. http://www.sunalbs.com/research/knowedge. 1995

[5] Miller G A. WordNet: A Lexical Database for English. Comm ACM, 1995. 39～41

[6] M.W. Berry, Z. Drmac, and E. R. Jessup. Matrices, Vector Spaces, and Information Retrieval. SIAM Review, 1999, 41(2): 335～362

[7] Richrdo Beeza-Yates, Berthier Ribeiro-Neto. Moderm Information Retrieval. Addison-Wesley Longman Limited, 1999. 51～76

[8] 陈治平,林亚平,童调生.基于 *N* 层向量空间模型的信息检索算法[J].计算机研究与发展,2002,10:83-87

[9] YANHongcan, LIU Xiaobin. Research on Web Articles Retrieval Based on N-VSM [C],ICICA2010,452-459

[10] YAN Hongcan,ZHU Xiaoliang. The Design and Realization of the Linux Browser based on Webkit[C],ICTM2009,188-191

[11] ZHU Xiaoliang, YAN Hongcan . Research and Application of the improved

Algorithm C4.5 on Decision Tree[C],ICTM2009,184-187

[12] 丁璇等.中文网页标引源主题表达能力的调查统计[J].大学图书馆报，2002,10(6)：391-407

[13] 贾玉祥,昝红英,范明.基于概率模型的网页相关度研究[J].北京：全国第八届计算语言学联合学术会议,2005：23-37

[14] 张校乾,金玉玲,侯丽波.一种基于 Lucene 检索引擎的全文数据库的研究与实现[J].信息检索技术 2005：34-38

[15] 孙西全,马瑞芳,李燕灵.基于 Lucene 的信息检索的研究与应用[J].上海：信息系统,2006：45-49

第 6 章　领域知识的描述与本体建模

自然语言一直是一种有效的表示知识和交流知识的有效方法，然而计算机网络环境下的知识库系统却不能用自然语言来表示知识，根本原因是它的二义性和缺乏一致性的结构。要实现计算机自动处理知识并对知识问题进行求解，必须准确描述知识，以某种一致化的结构存储和组织知识。根据本体论的基本观点，任何复杂事物都可以从中剥离出事物的本质，也就是事物的最小化描述，即本体。随着语义网技术的发展，基于可扩展标记语言(XML)的 DC(DublinCore)元数据在知识本体[1]的描述中越来越受到青睐。

6.1　本体

本体(Ontology)是近年来计算机及相关领域普遍关注的一个研究热点，作为一种能在语义和知识层次上描述信息系统的概念模型建模工具，已被广泛应用于知识工程、系统建模、信息处理、数字图书馆、自然语言理解、语义 Web 等领域之中。本体明确了领域内共同认可的概念术语，利用领域知识的语义模型表达了概念含义，并在内部层次当中规定了这些概念之间的关系，为知识表示以及获取奠定了基础。

20 世纪 90 年代以来，研究人员从各自的专业角度出发对本体的理论和应用进行了深入研究，取得了丰富的研究成果，本体理论与技术也随之日趋成熟。

6.1.1　本体的相关概念

本体的概念来自哲学领域，研究事物客观存在的本质，是对客观存在的系统的阐释或说明，重点关注客观存在的抽象本质。后来被用于计算机学

科并受到广泛关注，但是本体的定义在计算机领域并未得到统一，学者们相继给出了本体的不同定义，其中 Gruber 在 1993 年给出的定义最为流行："本体是概念模型规范化的描述"，而人们认可最多的是 Studer 对本体的定义，他认为本体是共享概念模型准确的形式化描述。这个定义包含四层含义：共享（Share）、概念化（Conceptualization）、明确性（Explicit）和形式化（Formal）。

（1）共享指本体中的知识是大家都认同的，是协商一致的；

（2）概念化指对事物的描述表示成一组概念；

（3）明确指本体中全部的术语、属性及公理都有准确的规范；

（4）形式化指计算机可以对其处理。

除了这两种定义，不同学者从不同研究背景出发，给出了本体的许多不同定义，尽管没有统一，但是定义都是从不同角度合理解释本体，这些定义相互补充并使本体应用范围扩大，而且定义的本质差别都不大，都认为本体是共享概念模型的反映，并对其进行了准确的描述。

Perez 等人认为本体可以按分类法来组织，他归纳出本体包含 5 个基本的建模元语（Modeling Primitive）。这些元语分别为：类（classes）、关系（relations）、函数（functions）、公理（axioms）和实例（instances）。其中，类也可以称作概念。

（1）函数和公理

概念的含义很广泛，可以指任何事物，如工作描述、功能、行为、策略和推理过程等等。函数是一类特殊的关系，在这种关系中前 $n-1$ 个元素可以唯一决定第 n 个元素。例如 Mother-of 关系就是一个函数，其中 Mother-of (x, y)表示 y 是 x 的母亲，显然 x 可以唯一确定他的母亲 y。

公理是对本体结构的描述，包括两种形式的公理：C 包含于 D，或者 C 等于 D，其中 C、D 可以是原子或者复合的概念或者属性。公理代表永真断言，比如概念乙属于概念甲的范围。实例代表元素。

（2）概念之间的关系

关系代表了在领域中概念之间的交互作用，如：子类关系（subclass-

of)。从语义上分析，实例表示的就是对象，而概念表示的则是对象的集合，关系对应于对象元组的集合。概念的定义一般采用框架(frame)结构，包括概念的名称，与其他概念之间关系的集合，以及用自然语言对该概念的描述。基本的关系有 4 种：part-of、kind-of、instance-of 和 attribute-of。Part-of 表达概念之间部分与整体的关系；kind-of 表达概念之间的继承关系，类似于面向对象中的父类和子类之间的关系；Instance-of 表达概念的实例和概念之间的关系，类似于面向对象中的对象和类之间的关系；attribute-of 表达某个概念是另外一个概念的属性。在实际的应用中，不一定要严格地按照上述 5 类元语来构造本体。同时概念之间的关系也不仅限于上面列出的 4 种基本关系，可以根据特定领域的具体情况定义相应的关系，以满足应用的需要。

总之，本体的目标是通过明确描述相关领域知识并对其进行统一表示和定义，使得大家共同理解该领域知识并在同一领域不同系统间可以共享，使人们更容易的获取领域知识。本体的定义也决定了它的两个不同特性：它对客观存在概念模型的反应使得其具有静态性；而它应用范围的广泛使得其具有动态性。因为领域不同、服务对象不同时本体的定义和构造也不相同。

6.1.2　本体分类

知识系统中存在不同种类的本体，根据不同的分类标准，可以将本体分为不同的种类。Mizoguchi 等人根据语境相关和语境无关的分类标准将本体分为：领域本体、常识本体、元本体和任务本体。而 Van Heijst 等人则将本体按概念结构的数量和类型及概念的主题两个维度来分类。若按第一个维度来分，本体可分为：术语本体、信息本体和知识建模本体；若按第二个维度来分，本体又可分为：应用本体、领域本体、通用本体和表示本体。人们一般根据本体的依赖程度将其划分为顶层本体、领域本体、任务本体和应用本体。其层次关系如图 6-1 所示。

(1) 顶层本体：顶层本体主要研究和描述最普通的概念，定义了最基本

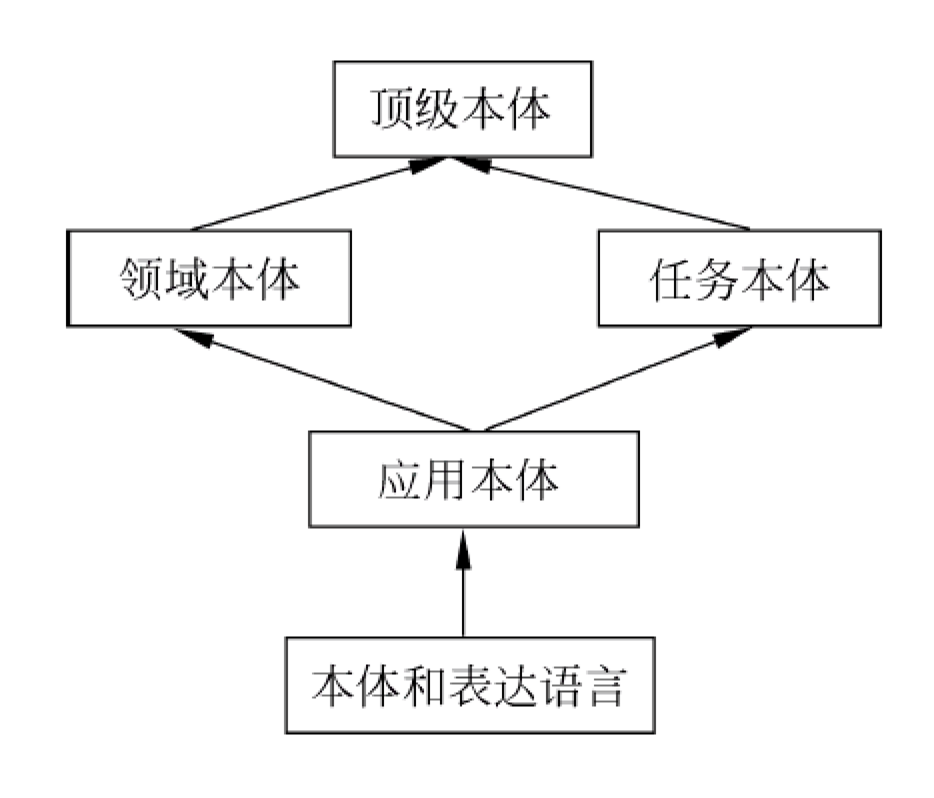


图 6-1　本体分类图

的概念类、属性及语义关系，如行为、时间、事物等，它不依靠于特定的问题与领域，共享范围广。其他种类的本体都是该类本体的特例。

(2) 领域本体：领域本体没有顶层本体那么广泛的共享性，是专业性的本体，对某个特定领域中相关的概念及概念之间的关系进行研究，如医学、农业等。这里的"领域"是根据本体构建者的需求来确立的，它可以是一个学科领域，可以是某几个领域的一种结合，也可以是一个领域中的一个小范围。

(3) 任务本体：任务本体与领域本体的处在相同的研究层次，它主要是定义一些通用的任务或推理活动，主要描述具体的任务中的概念及概念之间的关系，如培育、治疗等。

(4) 应用本体：主要用来描述在一定领域下为完成某项任务而需要的相关术语及术语间的关系。

从图 6-1 中可以看出各类型本体相互关联，领域本体和任务本体处于同一研究层次，它们都可以共享顶层本体中定义的通用概念，而应用本体同时依赖特定领域和特定任务或行为。

本体种类对任务和领域的依赖程度可以用如图 6-2 表示。

1999 年，Perez 和 Benjamins 在分析和研究了各种 Ontologies 分类法的基础上，归纳出 10 种 Ontologies：知识表示 Ontologies、普通 Ontologies、顶级 Ontologies、元(核心)Ontologies、领域 Ontologies、语言 Ontologies、任务 Ontologies、领域任务 Ontologies、方法 Ontologies 和应用 Ontologies。这种

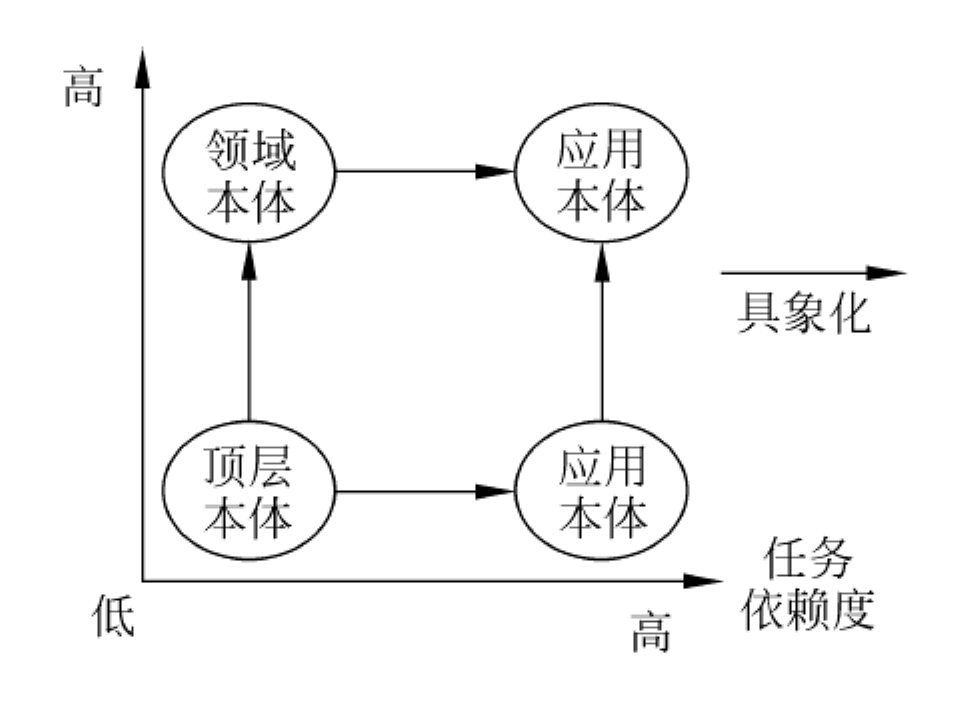


图 6-2　本体层次模型图

分类法是对 Guarin 提出的分类方法的扩充和细化，但是这 10 种 Ontology 之间有交叉，层次不够清晰。

6.2　领域本体建模

领域本体是用于描述指定领域知识的一种专门本体，它给出了领域实体概念化及相互关系领域活动以及该领域所具有的特性和规律的一种形式化描述[2]。有关本体论方法的研究和应用在知识工程、自然语言理解和知识表示等领域日益受到重视。领域本体（Domain Ontology）是专业性的本体，提供了某个专业学科领域中概念的词表以及概念间的关系，或在该领域里占主导地位的理论。

目前本体模型的研究已经进入到实际应用阶段。许多研究领域目前都建立了自己标准的本体，目前，Web 上有许多可重用的本体资源库，这使得诸多领域专家能够使用它们来共享和评注领域中的信息。构建领域本体要捕获相关的领域知识，提供对该领域知识的共同理解，确定该领域内共同认可的词汇，并从不同层次的形式化模式上给出这些词汇（术语）和词汇之间相互关系的明确定义。

本体工程方法包括特定领域的本体开发，如金融、化学、生物领域本体，还包括通常知识的本体库。领域本体是对特定领域内概念及概念间的关系的精确描述。领域本体的建立对于需要交换信息共享信息的人或者异构系

统来说，将有助于消除在概念和术语上的分歧，对于领域内的概念理解达成共识。其目标是捕获相关领域的知识，提供对该领域知识的共同理解，确定该领域内共同认可的词汇，并从不同层次的形式化模式上给出这些词汇间相互关系的明确定义。

领域本体是一个五元组，记作 $O=\{C,A,R,I,M\}$。其中，C 是概念集，指特定领域中属于概念的集合；A 表示属性集，主要用来表现概念自身的特征；R 是关系，指领域中概念间的相互作用；I 为实例集；M 是实例与概念之间的映射关系集合，该映射集将每个实例对应到其所属的概念下。

领域本体是建立在领域概念及概念之间抽象关系的基础上，不依赖于具体的软件存在，从而可以成为面向该领域的通用模型，因此具有极高的可重用性，方便在其之上进行开发和应用。

领域本体的构建方法主要有以下几种：

1. TOVE 法

TOVE 法，又称 Gruinger&Fox 评价法。TOVE 是指多伦多虚拟企业(Toronto VirtualEnterprise)，专用于构建 TOVE 本体(关于企业建模过程本体)(Fox 1995，Gruninger 1996)，由多伦多大学企业集成实验室(Enterprise Integ ration Lab)研制，使用一阶谓词逻辑进行集成。TOVE 本体包括企业设计本体、工程本体、计划本体和服务本体。TOVE 流程见图 6-3。

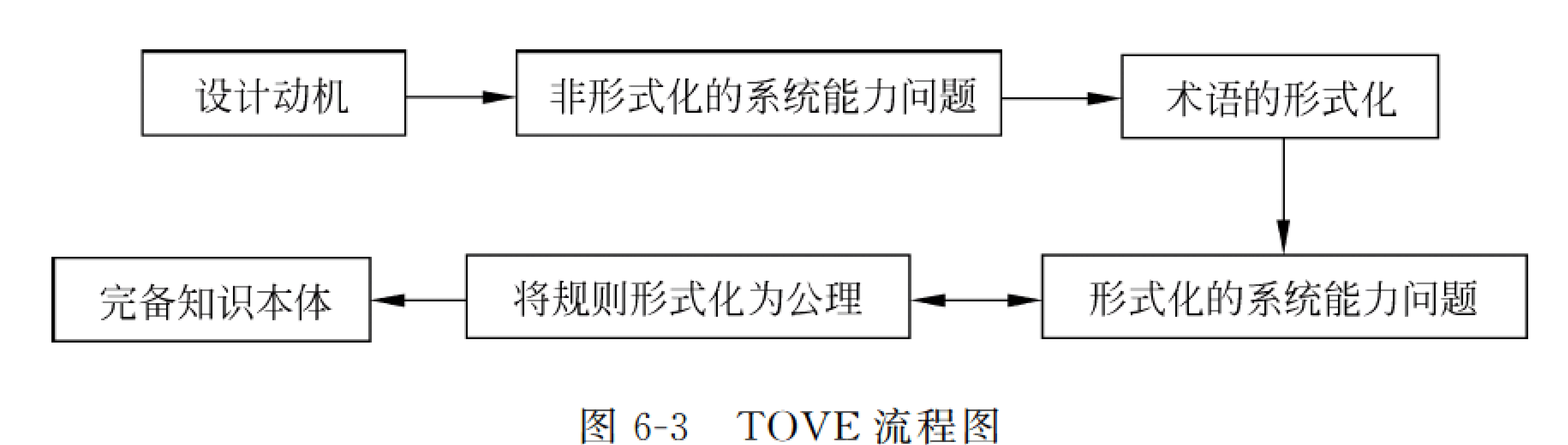


图 6-3 TOVE 流程图

2. METHONTOLOGY 法

METHONTOLOGY 法[3]，专用于构建化学本体(有关化学元素周期表的本体)(Fernandez 等 1996，Fernandez 等 1999)。该方法已被马德里大学

理工分校人工智能图书馆采用。它的流程包括：①管理阶段：这一阶段的系统规划包括任务的进展情况、需要的资源、如何保证质量等问题。②开发阶段：分为规范说明、概念化、形式化、执行以及维护五个步骤。③维护阶段：包括知识获取、系统集成、评价、文档说明、配置管理五个步骤。目前，用这种方法开发的本体有：(Onto) 2Agent，它是基于本体的 Web 代理，是使用参考本体作为知识源，在一定的约束条件下进行新知识获取的工具；化学本体(Chemical Ontology)，是基于本体的化学教育代理，允许学生在学习的基础上自测本身在该专业领域内所达到的水平。

3. 骨架法

骨架法(Mike Uscholddede & King，Usc95)，又称 ENTERPRISE 法，专门用来构建企业本体(ENTERPRISE ontology)，是有关企业建模过程的本体(Stader，1996)。建立在企业本体基础之上，是相关商业企业间术语和定义的集合，该方法只提供开发本体的指导方针。目前企业本体项目由爱丁堡大学人工智能研究所(AIAI-the Artificial Intelligence Applications Institute)及合作伙伴-IBM，Lloyd's，Register，Logica UK Limited 和 Unilever 共同承担。骨架法流程见图 6-4。

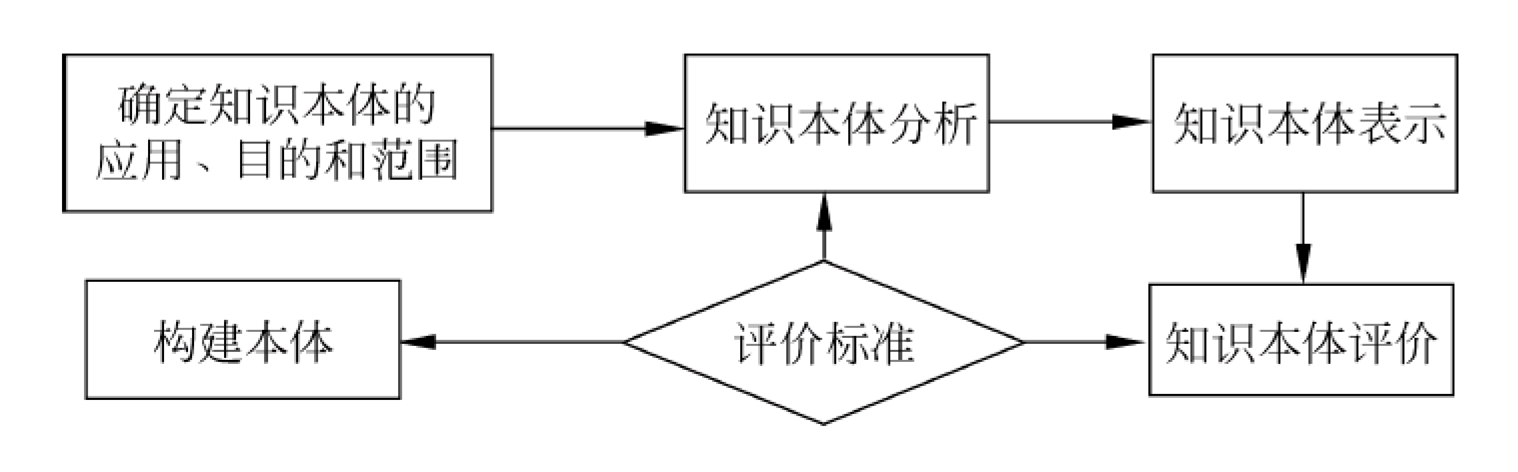


图 6-4　骨架法流程图

4. KACTUS 工程法

KACTUS 工程法(KAC96)是基于 KACTUS 项目而产生的，KACTUS 是指关于多用途复杂技术系统的知识建模工程(Modeling Knowledge about Complex Technical Systems for Multiple Use)，是欧洲 ESPR Ⅱ框架下的研发项目之一属于 ESPR Ⅱ-Ⅲ所支持的项目。支持 EXPRESS 和 Ontolingua

语言。①应用说明：提供应用的上下文和应用模型所需的组件；②相关本体范畴的初步设计：搜索已存在的本体，进行提炼、扩充；③本体的构造：用最小关联原则来确保模型既相互依赖，又尽可能一致，以达到最大限度的系统同构。

5. 七步法

斯坦福大学医学院开发的七步法，主要用于领域本体的构建。七个步骤分别是：①确定本体的专业领域和范畴；②考查复用现有本体的可能性；③列出本体中的重要术语；④定义类和类的等级体系；⑤定义类的属性；⑥定义属性的分面；⑦创建实例。

表 6-1 对七种本体构建方法进行对比，均允许在系统间进行操作，都提供知识共享和复用的机制。方法体系的成熟度依次为：七步法→METHONTOLOGY 法→TOVE 法→骨架法→KACTUS 法。

表 6-1 本体构建方法比较

项　　目	工程管理	需求分析	设计	执行	开发后期
TOVE 法	没有	有	有	有	没有
M ETHONTOLOGY 法	不全，没有建立工程环境	有	有	有	不全，没有操作支持、维护和训练阶段
骨架法	没有	有	没有	有	没有
KACTUS 法	没有	有	有	有	没有
七步法	不全，没有建立工程环境	有	有	有	不全

以上几种方法均允许在系统间进行操作，都提供知识共享和复用的机制。方法体系的成熟度依次为：七步法＞METHONTOLOGY 法＞TOVE 法＞骨架法＞KACTUS 法。

6.2.1 形式背景抽取

本体的应用、目的和范围，均是形式背景的核心内容，从某一领域的形式背景中抽取核心概念是本体构建最关键的一步。领域概念是在特定领域

内具有语义的词或短语的集合。从某种意义上讲，领域概念是领域知识在文本中的外在表现。领域概念的获取就是从领域文本集合中抽取最多能够代表该领域的概念集合，这个过程包括从领域文本中抽取术语集合，词性规范(同义词处理)以及领域概念的筛选和确定。

一个能代表领域知识的单词或短语可以由术语自动提取出，但是，这仅仅是本体概念的候选，这些候选包括了以下几种情况：可能是概念，可能不是概念，也可能是概念的一个片段，还有可能是一些在特定上下文中出现时才会出现的概念，但在出现的另外的语言环境中却不能称之为概念。

一词多义在自然语言是相当普遍的现象。如果直接把术语当作本体概念，同一个术语的关系既可为词义 A，又可为词义 B，因此所形成的概念语义关系可能非常混乱。概念具有一对一的关系，具有的是一个确定意义，而术语可以是一对多的关系，可能具有多个意义，因此术语通常不是概念。

一个术语可以代表本体概念，前提是可定义其：

(1) 内涵：形式化定义该概念描述的对象集，例如："船舶是水路交通工具"。这里"水路交通工具"是"船舶的"内涵，如采用 ILP 和 FCA 方法。

(2) 外延：该概念所要阐述的是对象集(实例)，即概念实例，例如："船舶"这一概念的外延包括货轮、渔船、客轮，以及其他形式的船舶。

(3) 词法实现：多语种同义词与术语本身，含有概念的同义词和多语种表示。

抽取概念定义是获取概念内涵的途径，又可定义为语义排歧，是对于那些语义多于一种的术语。这个过程可以通过学习确定它在特定的环境下取某一个语义。这也可表示在概念学习的基础上，各个术语都具有明确的定义(含义)。此时作为概念的术语才与其定义吻合，在此基础上建立的关系也进一步合理。

SSI(structure semantic interconnection)是现阶段语义排歧比较有效的、经典的算法。具体表述为对于一个术语集 T_{Set}，首先可以通过 WordNet 发现它们中的独义词，并将其从 T_{Set} 移入集合 I(I 表示已排歧的术语集合)。然后对于 T_{Set} 中的每一个术语 t，分别取它在 WordNet 中的每个语义 Si。利

用 WordNet 定义的各种关系,在一定深度范围内遍历术语 t 在语义 Si 时和 I 中各个术语的联系,并把联系程度按照一定方法量化。在术语 t 的各个语义都计算完成后,取与 I 联系最紧密的语义。如此循环,直至所有术语都被排歧或在最近的一次循环中没有术语被排歧。

(4) 实例抽取。

以语义分析,概念指的是对象的集合,本体实例为组成概念的成员。实例表示概念的具体个体,所谓实例学习就是在领域语料集中识别出属于某个概念的个体。本文对实例抽取采用了两种方法:一是实例关系抽取法,另一种是模式匹配方法。基于模式匹配方法的基本算法思路是首先定义一些模式,然后利用 ICT(中文语料)和 QTag(英文语料)对领域文集进行词性标注,最后应用模式匹配算法在领域文集中寻找实例。

本体实例获取是通过一定的学习方法和工具,利用已知本体库从 Web 语料库或其他知识源中,以自动或半自动地方式发现领域概念实例和属性关系实例。它可以用四元组(R,O,A,I)描述,其中:R 为 Web 语料库或其他知识源;O 为领域本体库,O 中至少含有概念集合 C 和概念的属性关系集合 P;A 为本体实例获取算法;I 为领域概念实例和属性关系实例的学习结果。

这里的模式是指那些以字符序列形式出现,且不能将这些字符分成相互独立的关键字的一种数据。模式处理不但包含了数据插入、搜索、删除、替换等基本运算外,而且还有其特有的字符串查找和运算—模式匹配:给定一个长度为 n 的字符串(通常称为正文)Text$=t_1t_2\cdots t_n$,以及另一个长度为 $m(m\leqslant n)$ 的字符串(通常称为模式)Pattern$=p_1p_2\cdots p_n$,这里 $t_i(1\leqslant i\leqslant n)$ 和 $p_j(1\leqslant j\leqslant m)$ 均是字典表中的字符,要求查找出模式 Pattern 在正文中首次出现的起始位置(即下标)。一旦在正文中找到一个模式,则可认为它发生一次匹配。

朴素模式匹配算法是比较普遍的模式匹配应用。其原理是将匹配问题转化为以模式为关键字查找问题,具体是将长度为 n 的正文 T 划分为 $n-m+1$ 个长度为 m 的子字符串(简称字串),检查比较每个这样的子串是否与长度为 m 的模式 P 相匹配。

本阶段需要明确领域本体构建的目的、范围用途和使用者。从表面上看,领域本体的构建是为机器服务的,其最终目的还是为用户提供信息服务。采用和软件开发过程类似的办法,在本体构建的初期,首先了解其应用的具体背景和需求。考虑到领域知识的深度和广度以及关系的复杂程度,所以采用本体时尽可能覆盖领域内的所有知识。但一味地扩大本体的范围会造成工程的复杂度和成本的剧增,甚至会造成工程的失败,这是需要注意的。根据领域专家提供的知识和实际需求控制知识范围,尽可能使本体范围在较小的情况下满足要求。本体开发者在实践中肩负完成形式化知识,使编码成为计算机可处理语言的任务,所以本体开发者需了解本领域的基本知识,包括特点、规则以及技术方法,以此作为与领域专家交流合作的基础。

6.2.2　领域属性概念定义

在建立起的本体框架中,概念需要被精确定义,尽可能准确而精简的表达出领域的知识。除了概念,还要定义概念之间的关系。这些关系不仅仅涉及同工作领域的概念,不同工作领域的概念也可以相关,只是这些关系总是属于某一个工作领域。在创建的概念中,很大部分属于类,对类及其层次、类的属性、属性值等要重点定义并创建类中的实例。

1. 定义类及类的层次

在创建的概念中,很大一部分属于类,而对类的层次的定义有以下 3 种方法:①自上向下法。先定义领域中综合的、概括性的概念,然后逐步细化、说明;②自下向上法。先定义具体的、特殊的概念,然后对这些概念泛化成综合性的概念;③混合法。混合使用自上向下法与自下向上法,先建立那些显而易见的概念,然后分别向上与向下进行泛化与细化。这 3 种方法各有利弊,采用哪种方法取决于开发人员对领域认识的角度。如果设计人员对该领域有自上向下系统的认识,第①种方法比较适合。一般来说,混合法比较适合大部分设计人员。

2. 定义类的属性

仅仅通过类是不足以确切地描述一个领域的，因此，一旦定义好了类就要描述这个类的内部结构。已经从步骤②中创建的概念中选择了类，大部分剩下的概念可能成为这些类的属性，确定属性是用来描述类。在本体模型中，属性往往具有以下一些特征：①本质特征，如颜色、运行速度；②外来特征，如产地、厂商；③组成部分，如材料；④个体间关系。子类继承父类所有的属性，因此属性应当被尽可能定义在通用类中。

3. 定义属性值

属性值既可以是一个数值也可以是一个类。我们将属性值视为一个类，称为属性类。属性类也有属性，通过这些属性来说明取值类型、值个数及有关值的其他特征。

4. 创建实例

创建概念类中的个体实例，选择概念类并创建该概念类的实例。

使用描述逻辑表示领域共享语词库。本体建立者在本体的领域和范围内进行概念化的首要任务是建立领域共享语词库，并收集所有有用的和潜在有用的领域概念及语义、属性和实例等。建立领域共享语词库，首先通过收集领域信息充分了解领域知识。信息来源可以包括专家、书籍、网络，甚至是其他的本体。它是对领域本体的细化过程，叙词表为共享语词库的建立提供了必要的条件。

共享语词库的建立是根据叙词表对每一个涉及的分类领域进行语义描述[2]。本体建立者应当写入下列词典项：

1 class name；2 synonym o f the class；

3 properties of the class；4 relations o f the class。

例如：农产品原料的类名；该类农产品原料的不同叫法；该类农产品原料的属性：定购数量，即两企业间交易该类型原料的数量；价格，即两企业

间的交易价格等等；农产品原料的部分重要的关系：所属类别，即描述了该原料所属的类别以及和其他原料之间的关系；原料供应企业，即描述了该原料是由那个企业所生产的等等。

用 OWL 描述本体，在描述逻辑的基础上建立概念分类层次，然后定义类的属性和创建类的实例。由于要涉及 RDF 的概念，先简述几个概念：RDF 的主要元素是根元素、<RDF>，以及可识别资源的<Description>元素。<rdf RDF>是 RDF 文档的根元素，定义了一份 XML 文档将成为一份 RDF 文档。此外提及了 RDF 名称空间(namespace)，< rdf Description>元素通过 about 属性识别资源。

6.2.3　本体建模工具 Protégé

手工构建本体是一个耗时、昂贵、高度技巧化、非常复杂的过程，通常需要周而复始地反复修改编辑。经过近十几年发展，本体编辑工具已经比较成熟。在这个发展过程中因本体描述语言的不同出现两类本体开发工具。

第一类是基于人工智能(AI)的特定描述语言的本体开发工具，包括 Ontolingua（基于 Ontolingua 语言)，OntoSaurus(基于 LOOM 语言)，WebOnto (基于 OCLM 语言)等。

第二类是不依赖于某种特定描述语言的本体构建开发工具，利用基于组件的结构进行功能的使用与扩展，组件的结构指整合的开发环境或组件的集合，能对基于 Web 的本体描述语言格式(例如，XML，RDF(S)，DAML＋OIL 语言)进行导入/导出操作，包括 OntoEdit、Protege 系列等。

不同的本体开发工具应用于不同的专业领域。在这些工具中，被使用最广泛、最受关注的是斯坦福大学医学情报研究组开发的本体构建编辑工具——Protégé(http://protege. Stanford. edu)。该工具采用 Java 编写，可以免费下载，其界面与普通的 Windows 应用程序风格一致，由于其开放性和兼容性备受瞩目，成为目前本体编辑的首选工具。

Protégé 有很多其他编辑工具所不具备的优点：

① 是一个开放资源，允许用户二次开发，目前拥有最多的注册用户；

② 不断有新版本推出,不断增加新功能;

③ 系统的可扩展性好,支持下载安装或者自行开发插件,拓展 Protégé 的功能;

④ 能够以多种方式存储本体,包括多种数据库格式和纯文本格式,可以适应不同需要;

⑤ 支持多种本体表示语言输出,包括 XML、FRFS、OWL 等;

⑥ 界面简单友好,易于用户学习和操作;

⑦ 拥有众多的可视化插件供用户选择,概念关系一目了然;

⑧ 支持中文编码。

当然,Protégé 在设计上也存在着一定程度的不足:

① 一次只能打开一个本体:有些大型本体的编辑需要参考和引用已经存在的本体,这样的方式会影响大型本体的工作效率;

② 不支持协同开发:目前的 Protégé 只有客户端,没有服务器,不支持协同开发,尤其是大型本体的开发,需要多人的合作,大大影响本体的编辑效率;

③ 不能进行批量处理:在输入实例的过程中,不能以文本的形式将同类型的内容一次性导入,输入比较繁琐;

④ 运行速度比较缓慢,编辑效率不高,启动需要占用大量的内存资源;

⑤ 部分图形化显示工具不支持中文本体显示,如 OWLViz。

在 Protege 本体编辑器中,本体结构以属性的层次目标结构显示,用户可以通过点击相应的项目增加或编辑类、子类、属性、实例等,使用户在概念层次上设计领域模型[4],所以本体工程师不需要了解具体的本体表示语言。Protege 支持多重集成,并对新数据进行一致性检查,并且具有很强的可扩展性。同时文件输出格式可以定制,可以将 Protege 的内部表示转化成多种形式的文本格式,包括:XML,RDF(S),OIL,DAML,DAMI+OIL,OWL 等系列语言。具体的操作方法如下:

① 建立新的本体项目。运行 Protege4.1,选择新建一个本体,根据提示,分别完成命名空间、存放位置以及选择本体的描述语言等操作。

② 定义类及类的层次。利用 Class 标签，为完成分析的领域本体定义类及类的层次。在 Protege4.1 中可以对类进行设置，如父类、不相交类等。

③ 设置类的对象属性。利用 Object Properties 标签对类间的语义关系进行描述，以及属性的定义域（domain）、值域（range），增加对类间语义关系的限制。如把属性设置为：

1）翻转属性（Inverse Property）

如果 P1 被声明为 P2 的翻转属性，那么如果 X 通过 P1 关联到 Y，那么 Y 通过 P1 关联到 X。如果 hasChild 是 hasParent 的翻转属性，Deborah hasParent Louise，那么我们就能够推理出 Louise hasChild Deborah。

2）传递属性（TransitiveProperty）

如果（x，y）是传递属性 P 的一个实例，（y，z）也是传递属性 P 的一个实例，那么（x，z）是传递属性 P 的一个实例。如果 ancestor 被声明为传递的，（Sara，Louise）是他的一个实例，（Louise，Deborah）也是他的一个实例，那我们就能够推理出（Sara，Deborah）是他的一个实例。

3）对称属性（SymmetricProperty）

如果（x，y）是对称属性 P 的一个实例，那么（y，x）也是它的一个实例。被声明为对称的属性不能有任意的 domin 和 range。Friend 可以被说成是一个对称属性，如果 Frank 是 Deborah 的 Friend，那我们可以推断出 Deborah 是 Frank 的 Friend。

4）为属性添加约束（Restrictions），包括：

（1）allValuesFrom 约束

该约束将一个属性的取值和一个 class 相关。也就是说，如果一个 class 的实例通过这个属性和另外一个 individual 相关，那么后一个 individual 则能够被认为是该越是类的一个实例。

Class Person 有一个属性 hasOffspring，该属性被约束在 allValuesFrom 上取值为 Person 类。这就是说如果 Person 的一个实例 Louise 通过属性 hasOffspring 和另一个 individual Deborah 相关，从这一点我们能推断出

Deborah 是 Person 的一个实例。这种约束允许 hasOffspring 属性被其他 class 使用,例如被 class Cat 使用,从而做出相应的约束。

(2) some ValuesFrom 约束

和上面类似,该约束也将一个属性的取值和一个 class 相关。只不过此时要求该属性的取值至少有一个是该 class 类型的。

5) 设置数据属性。利用 Date Properties 标签来描述某类本身所具有的特性,如人的性别,年龄等,并对取值类型进行定义,如定义日期的取值范围为时间类型的值。

6) 向类添加实例。利用 Individuals 标签为已经设置好的类添加实例,赋予相关属性的值。

在完成上述步骤之后,可以调用集成于 Protege 中的相关推理机,进行基于公理的推理,系统会自动提示非法的字符及层次、限制等关系,检验本体的逻辑结构,便于进一步修改完善本体。

6.3 XML 数据到 OWL 本体的转换

XML 是互联网中的 W3C 标准文档格式,用于写和交换信息,目前 XML 在电子商务中一直是普遍接受的数据交换形式。XML 覆盖了语法层,但是缺乏对概念化共享的有效支持。幸运的是,OWL 是一个 W3C 标准模式,用于描述语义和对互联网上的信息推理,对语义协调的技术有着很好的支持。OWL 通过使用类、属性和实例在分布式互联网的环境中支持领域知识的表示。然而,OWL 和 XML 虽然很相似,但毕竟是基于两种不同的表现形式,因而应用时需要互相转换。

对 XML 数据,元数据是 XML Schema,因为它定义了 XML 文档的标签和结构,描述了 XML 实例文档中数据的类型等信息,其语义和数据是在一起的,因此,在从 XML 到 OWL 的转换过程中,一个语法正确的 XML Schema 的存在可以优化对 XML 文档的操作。但最新的研究表明,实际应用中的许多 XML 文档要么没有相应的 XML Schema 的存在,要么基于一个

语法错误的 XML Schema。为了获得更为完备的 OWL 文档，我们首先需要从给定的 XML 文档中自动生成一个 XML Schema。

对于 XML Schema 的提取的研究大多利用了 DTD 的内容模型生成方法。一些具有 XML Schema 提取功能的软件或工具，如 Trang、Atoval XMLSpy、Stylus Studio，其生成的 XML Schema 在表达能力上都是等同于 DTD 的。这里介绍文献[5]提供的 Schema 挖掘算法通过语境化能力来获取上下文相关性的相关信息，实验表明，与同类方法相比，此算法具有更好的可扩展性和表达能力。在接下来的部分将介绍从给定的 XML 样本集中获取该样本集的 XML Schema 的推理算法[5]。

6.3.1　相关定义

在介绍算法前，先对相关定义做一个简单介绍。

1. XML 片段

对我们来说，一个 XML 片段就是一个如<al>fl</al>w<an>fn</an>形式的元素队列（可能为空），其中 al，…，an 是元素的名称，fl，…，fn 是 XML 片段本身。特别的，忽略属性（因为他们可以直接添加上去）和数据的值。

通常将<a>简写为<a/>。此外，假如 f 是一个 XML 片段，用 path(f)来表示从 f 的根节点开始的所有的标签路径集。以图 6-5 所示的 XML 片段为例：path (f)路径包括空路径 A、路径 store、路径 store order、路径 store stock、路径 store order customer 等。我们用 strings ＜ f，p＞来表示 XML 片段 f 的一级目录下的所有的元素名字字符串的集合。仍然以图 6-5 所示的 XML 片段为例：

strings(f,λ)＝{store}，strings(f，store)＝{order order stock}，strings (f，store order) ＝{customer item item，customer item，从 XML 片段图 6-5 中的第一个 orde：元素得到元素名字字符串 customer item item，从第二个 order 元素得到名字字符串 customer item。对于以叶子节点结束的路径如 store order customer name，strings ＜ (f，p) ＝λ。

```
<store>
  < order>
      <customer> <name/> </customer>
      <item> <price/> </item>
      <item> <price/> </item>
  </order>
  < order>
      <customer> <name/> </customer>
      <item> <price/> </ltem>
  </order>
    <stock>
      <item>
        <id/>
        <supplier> <name/> </suplier>
      </ltem>
    </stock>
</store>
```

图 6-5 XML 片段样例

2. XML Schema 定义

W3C 规范将 XSD（XML Schemas Definition，XML 结构定义）定义为一个类型定义的集合 D，我们从具体的 XML 表述中抽象出的 XSDs 形式如下所示：

$\text{store} \to \text{order}[\text{order}]^{*}, \text{stock}[\text{stock}]$

它将类型名字通过成对的元素名称 a 和类型名字 t 将类型名称映射到正则表达式 a[t]。直观地说，这种特殊的类型定义将<order>fl</order>…<order>fn</order>，<stock>g</stock>形式的 XML 片段规定为类型 store，当 $n \geq 0$ 时，fl，…，fn 是类型 order 的一个 XML 片段，而 g 是类型 stock 的一个 XML 片段。任何一个出现在 XSD 的类型定义的右边的类型名称必须在 XSD 中被定义，任何一个类型名字只可以被定义一次。

尤其需要注意的是：W3C 规范的“元素声明一致性（Element Declaration Consistent）”要求在同一类型定义中多次出现的相同元素名字必须类型相

同。因此,类型定义 store→ order[order] * , stock[stock]是合法的,但形如 person→ (person[male]+person[female])$^+$,由于 person 既在类型 male 中出现,又在类型 female 中出现,所以它是非法的。当然,在不同的类型定义中同一个元素名称可以以不同类型出现(这正是让一个元素的内容模型依赖它的上下文所产生的能力)。例如,图 6-6 中所示的 XSD 描述即是对图 6-5 中所示的 XML 文档的定义,需要注意的是,order item 和 stock item 分别用 $item_1$ 和 $item_2$ 来进行定义。

由于"元素声明一致性"的约束,类型 t 的类型定义中的每一个元素名字 a 通过唯一的类型 T(t, a)与类型 t 相关联。我们以图 6-6 所示的 XSD 为例:τ(root,store)=store, τ(store, order)=order,τ(store, stock)=stock,τ(order,item)=iteml,τ< (iteml,id)=emp,τ<(stock,item)=item2。然后通过移除 t 的定义中的所有的类型名字,用只包含元素名字的 ρ(t)来表示 XSD 中的正则表达式。则对图 6-6 中的 XSD 有: ρ(roof) = store; ρ<(store)=order * , stock; ρ (order)=customer, item+; ρ (person) = name, email+等。于是,把一个 XSD D 简化为一个二元组:①被定义的类型集 T,②映射 τ,③映射 ρ。以 order 的类型定义为例,我们用一对 a[T(order, a)]来替换正则表达式ρ(order)=customer,item+中的元素名称 a,从而可以轻易得到图 6-6 中所示的 Order 的正则表达式。

```
root→store[store]
store→order[order] * ,stock[stock]
order→customer[person ], item[item]+
person→name[emp]
item1→price[emp]
stock→item [item2]+
item₂→ :— id[emp], supplier[person ] + item [item2]+
emp→λ
```

图 6-6　图 6-5 中所示 XML 文档的 XSD 描述

3. 语境化能力(Contextual power)

从上面的验证算法可以看出,f ∈F(D,t)中的任何元素的内容模型完全

由从 XML 根节点到该元素的标签路径来决定。只有在当每一个 fi 都被 w. r. t. τ((t, ai)验证的情况下,f=<a1>f1</a1>…<an>fn</an>才为类型 τ。只有在 fi=<bl>gl</bl>…<bn>gn</bn>且每一个 gi 都被 w. r. t. $\tau(\tau$ (t, ai),bj)验证通过后 gi 才是类型 τ。于是这一理论将引导到另一种验证方法,从而形成我们的推理算法的基础。对于一条路径 p=ab…c,τ(s, p) →t 表示 τ (…τ(s,a) b),…c)已经被定义并且等同于 τ。假设 L(r)是表示通过正则表达式 r 匹配过得所有的字符串的集合。

4. 定位(locality)

有研究发现,98%的 XSDs 中的某个元素的内容模型实际上并不依赖于从根节点到该节点的整个的标签路径,而只是依赖于这条路径上的最后 k 个元素的名称,通常 $k \leqslant 3$。这种 k-local XSDs 的定义如下:假设 $p|k$ 表示由路径 p 的最后 k 个元素名字形成的路径(如果 length(p) $\leqslant k$,则 $p|k=p$)。当 $p|k=q|k$ 时,我们认为路径 p 和路径 q 是 k-等价的。特别地,当 length(p)$<k$ 时,p 只与它本身 k-等价。

5. 单一事件

研究发现,99%以上的 XSDs 的正则表达式中,每个元素名称都只出现最多一次。有定义:如果一个正则表达式中的每一个元素名字在该表达式中最多出现一次,那么该正则表达式是单一事件。如果一个 XSD 只包含单一事件的正则表达式,那么该 XSD 是单一事件。

例如:customer, item^{+}和(school+ institute)$^{+}$都是单一事件,但是由于 id 在 id,(qty + id)中出现了两次,故 id 和(qty, id)不是单一事件。"单一事件正则表达式"为 SORE,"单一事件 XSD"被简称为 SOXSD。

6.3.2 Schema 挖掘算法

基于 Schema Miner Method 的 Schema 挖掘算法是一种启发式的算法,在生成 XML Schema 的过程中主要关注两点:元素类型和无序序列。此

外，Schema 挖掘算法还可以定义具有不同名字但结构相似的相同类型的元素以及类型之间的继承关系。

Schema 挖掘算法通过以下三步生成给定 XML 文档的 Schema：首先根据 XML 文档元素的类型进行聚类分析；然后将聚类中的所有字符串推导出一个有穷自动机；最后将得到的有穷状态机转换为 XML Schema。

1. 聚类元素

XML Schema 语言是一种基于类型的语言，它的每一个元素都必须有一个类型。Vosta 提出的 Schema Miner method 主要侧重于推导出具有相同名字但结构不同的不同类型的元素，该算法根据元素的名字进行聚类，当同名元素具有不同的机构时对这些聚类进行拆分。这里的 Schema 挖掘算法在此基础上增加了定义结构相似但名字不同的相同类型的元素的功能。

定义：如果 XML 元素的一个集合 Ct 具有相同的 XSD 类型，那么我们称 Ct 为类型 T 的聚类。XML 元素 E 的字符串是 E 的子元素按特定顺序排列的一个名称字符串。类型串 St 是指所有的 E∈Ct 的字符串集。

算法 6-1 的思想如下：首先根据元素的语境对元素进行聚类，由于 XML Schema 不允许相同语境中的同名元素类型不同，因此，同名而且绝对路径相同的元素将具有相同的类型。在元素根据其语境进行聚类后，算法将判断不同的聚类是否足够相似，如果两个聚类足够相似，那这两个聚类将被归并为同一类型。算法 6-2 中给出的树编辑距离算法将被用来处理两个聚类之间的相似度。

算法 6-1　类型归并算法(mergeTypes)

```
输入：一个聚类集 C
输出：经过类型归并后的聚类集 C
for each pair T1, T2 in C do
distance := treeEditDistance(T1.representant, T2.representant);
    if distance < MAX_DISTANCE then
        merge(C, T1, T2);
    end if
end for
return C
```

算法 6-2 树编辑距离算法(treeEditDistance)

```
输入: 一对 XML 树 T1, T2
输出: 整数 0-100 范围内的树编辑距离
1: distance :=nodeDistance(T1 .noot, T2 .noot, Tl. root, T2 .root, true);
2: return (distance * 100)/(T1 . size+T2.size};
```

如果没有聚类被归并,则需要做 $n^2/2$ 次比较,其中 n 是初始状态下聚类的个数。经过归并后,比较的次数可能会迅速下降。

算法 6-3 节点距离算法(nodeDistance)

```
输入: node1, node2, rood, root2,布尔类型的递归
输出: node1 与node2 之间的距离(整数类型)
nodesList1 :=sortChildNodes(node1);
nodesList2 :=sortChildNodes(node2);
distance :=0
while nodesList1.hasMoreElements AND nodesList2.hasMoreElements do
    child1=nodesList1 .actual;
    child2=nodesList2.actua1;
if childl :name==child2:name then
distance+=nodeDistance(child1,child2, rood1,root2, recursively);
nodesList1 .next;
nodesList2. next;
else if childl. name<child2. name then
        if recursively AND node1.name==node2.name then
            subtreeDistance=subtreeDistance(child1;rood;root2);
            topDistance=topDistance(child1,node2, root1,root2);
            distance+=min(subtreeDistance, topDistance);
          else
          distance+=child1. size;
        end if
    nodesList1.next;
else
    if recursively AND node1.name==node2.name then
        subtreeDistance=subtreeDistance(child2; root2; roort1);
        topDistance=topDistance(child2, nodel, root2, root1);
        distance+=min(subtreeDistance, topDistance);
    else
        distance+=child2.size;
    end if
```

```
        nodesList2. next;
    end if
end while
while nodesList1.hasMoreElements do
      child1 =nodesList1 .actual;
if recursively AND node1.name==node2.name then
      subtreeDistance=subtreeDistance(child1, root1, root2);
      topDistance=topDistance(child1, node2, root1, root2);
      distance+=min(subtreeDistance, topDistance);
      else
            distance+=child1.size;
        end if
    nodesList1.next;
end while
while nodesList2.hasMoreElements do
      child2 =nodesList2.actual;
      if recursively AND node1..name==node2.name then
            subtreeDistance=subtreeDistance(child2, root2, root1);
            topDistance=topDistance(child2, node1, root2, root1);
            distance+=min(subtreeDistance, topDistance);
        else
              distance+=child2.size;
        end if
      nodesList2.next;
end while
return distance;
```

算法 6-4　子树距离算法(subtreeDistance)

```
输入: 要搜索的子树, root1 子树的根 ,root2 子树的根
输出: 递归所得到的最小距离
1: elementsList :=findInSubtree(child, root2);
2: for each element in elementsList do
3:    min :=min(min, nodeDistance(element, root2, root1,root2, false));
4: end for
5: return min;
```

算法 6-5 顶部距离算法(topDistance)

```
输入: 要搜索的子树, 开始搜索的节点, rood 子树的根, root2 子树的根
输出: a minimum distance for the recurse
1: elementsList := findInTopPath(child, node, root2);
2: for EACH element IN elementsList do
3: min :=min(min, nodeDistance(child, element, rood, root2, false));
4: end for
5: return min;
```

2. 类型的继承机制

Schema 挖掘算法将类型的继承机制置于类型归并阶段的后面,通过对类与类之间的比较,Schema 挖掘算法可以推出子类与超类的关系。

3. Schema 的生成

在聚类分析并标记出类与类之间的继承关系后,将从元素类型中推导出 Schemao XML Schema 的内容模型。可以通过一个正则表达式或一个与之等价的确定的有穷自动机来表示。在 Schema 挖掘算法中,内容模型被表示成一个模式自动机,这个模式自动机是有穷自动机的一个特殊形式。因此,生成 Schema 的问题被转换为寻找可以接受类型 T 的字符串集 ST 的模式自动机的问题。

定义:一个模式自动机(Schema automation,SA)是一个扩展的有限状态自动机。它可以表示为一个六元组 $(\Sigma, S, Sx, S0, \delta, F)$,其中:$\Sigma$ 是一个输入字母表(一个有限的,非空的元素名称集);S 是一个有限的基本状态的集合;Sx 是一个有限的扩展状态的集合;$S0$ 是一个初始状态,$S0 \in S \cup Sx$;δ 是状态转换函数:$S \cup Sx \times \Sigma \cup \{\lambda\} \rightarrow S \cup Sx$;$F$ 是终态集,$F \in (S \cup Sx)$。

模式自动机作用与确定自动机相似。它们之间唯一的区别是:模式自动机另外包含有一个扩展状态集。扩展状态 S_x表示子自动机 SA_x将接受部分的输入字符串。在到达 S_x后,SA_x将对输入继续进行处理。子自动机将

处理尽可能多的符号并将处理结果返回给父处理机。每一个扩展状态都有一个与之相对应的帮助状态。帮助状态是一个只有一个切入点的基本状态，它只能从扩展状态来切入。当子自动机将处理结果返回后，自动机将根据切入点进入到帮助状态。

模式自动机和有限自动机一样具有强表达能力。它可通过入边转换为有限自动机。扩展状态只有一个切入点，这个切入点将被间接地过渡到子自动机的初始状态。子自动机的所要终态都有一个额外定的切入点以切入到初始切入点的目标状态。有限自动机的处理过程如下所示：如果对当前字符有一个切换则进行切换；如果当前字符存在一个切入点，那就进入切入点；当对当前字符既不存在一个切换，也不存在切入点，则判断当前状态是否为终态，如果是终态则返回值为真，否则为假。通过归并使用入边进行连结的状态，新创建的带入边的有限自动机可以另外被转化为没有入边的有限自动机。

下面是在自动机中被表示出来的几种扩展状态：

(1) 继承状态：当某个类型是通过另外一个类型继承而来的时候继承状态将会被用到。其中的超类型是通过超类自动机来表示的。根据规则，子类的内容模型是超类的内容 i 型和定义在继承状态中的内容模型的一个串联。继承状态被用来表示超类自动机并被子为初始状态置于子类自动机中。这种构造保证了超类的内容模型和定义在其他自动机的内容模型之间的串联性。

(2) 序列状态：排列状态被用来表示 XML Schema 中的 all 原子类型。

(3) 群组状态：群组状态表示的是一个任意的自动机。群组状态被用来表示 group T 子类型，group 原子类型是对全局定义原子类型的一个引用。这种状态允许自动机在多个地方重用自动机架构，从而减少被定义的状态的数量。

4. SA 构造器

在得到类型聚类 Ct 后，需要为给定的输入串 St 构造一个 Schema 自动

机。通过算法 6-6 生成模式自动机，该自动机通过一个类型 T 来进行初始化。如果 T 是一个扩展状态，那么它的初态将被创建为一个初始状态并为之添加一个辅助状态，然后所有的输入字符串被添加到模式自动机。算法中的 nextState 方法将为所有的字串找到合适的转换。这里假设扩展状态的字符串可以被超类自动机接受，如果某字符串不能被接受，那么从该字符串生成模式自动机将没有扩展状态。

算法中应用了一个优化方法，即当输入字符串中包含了一个长度大于用户自定义的固定值 REPETITION 的序列时，将自动创建一个多重循环。该优化方法置于模式自动机构造器中，它可以显著地减少模式自动机的数量。

算法 6-6 Schema 自动机生成算法(createSchemaAutomation)

```
输入: 用于创建Schema 自动机的类型T
输出: Schema 自动机
schemaAutomaton=newSchemaAutomaton( );
  if T. isExtended then
    schemaAutomaton. initia1State=extensionState(T. parent);
    helperState=state( );
    schemaAutomaton. initialstate. createLambdaTransitionTo(helperState) ;
  else
     schemaAutomaton. initia1State=state( );
  end if
  for all string IN T. inputStrings do
       actua1State=schemaAutomaton. initialState;
  while string has more letters do
        letter=string. next;
        if string has repetition of letter then
           actualState. createLoopFor(letter);
        else
           nextState=actua1State. nextState(string);
           if nextState==null then
               nextState=state();
               actualState. createTransitionTo(nextState, letter);
           end if
              actua1State=nextState;
```

```
        end if
        end while actualState:final=true;
    end for
return schemaAutomaton
```

XML Schema 的生成过程如下：

在模式自动机创建完成后，把模式自动机转化为 XML Schema 的内容模型定义。由于所有的原子类型的内容模型可以被表示为一个带操作符 & 的正则表达式。因此，将模式自动机转化为 XML Schema 内容模型的过程等同为将 FSA 转化到一个正则表达式的过程。算法 6-7 给出了从模式自动机到 XML Schema 的内容模型的转化过程。首先，所有的终态将通过一个入边连接一个超终态，然后除初态和超终态之外的所有状态都将被一个表示该状态的所有后继结点、循环结点和前驱结点的新节点取代。

算法 6-7　正则表达式生成算法(SAtoregex)

```
输入: Schema 自动机A
输出: 经计算后的正则表达式
superFina1State=state( )
  for all state IN A. finalStates do
    state. createLambdaTransitionTo(superFina1State);
  end for
while A:states. size>2 do
  state=A. getLowestWeight();
  state. collapseParallelTransitions();
  loop=state. getLoop
  for all pair of backTransition and transition do
    regex=createRegexFor(backTransition, loop, transition)
    A. addTransition(backTransition. from, regex, transition. to)
  end for
    state :removeAllTransitions();
    A. removeState(state)
end while
  A. initialState:collapsePara11e1Transitions);
  loop=A. initialState. getLoop
  transition=A : initia1State. getTransitionToSuperFina1State();
  return createRegexFor(loop, transition);
```

6.3.3 生成 OWL 模型

XML 只能提供用户一个可以被应用自动读取的格式，而不能进一步表示数据的语义。尽管 XML 的灵活性使用户能够通过自定义的标签来快速、便捷的描述任意的内容，但计算机并不能理解这些标记的含义。虽然 XML 允许用户通过 XML Schema 来定义这些标记的集合，同时 XML Schema 也为 XML 文档提供了若干约束机制以限定 XML 文档中的标记及这些标记之间的结构关系，但 XML Schema 的语义仍然是隐含的。XML Schema 的元素的含义，要么由用户根据元素的名称（根据自然语言描述）去推断，要么通过另外的文档来进行描述，XML Schema 并不能对其所隐含的语义给出任何解释。

Semantic Web 用本体来描述领域知识的重要概念及各概念之间的关系。在层次化的语义网体系结构中，本体层主要用于提供语义支持，因此本体的构建是实现语义网的关键所在。本体是用来描述某个领域（领域本体）甚至更广范围（通用本体）内的概念及概念之间的相互联系，使得这些概念和联系在共享范围内有着明确的唯一的定义，以达成一种共识，从而使得人和计算机之间可以进行交流。通过由本体描述的语义文档，使得计算机可以理解和自动处理文档。

但是，本体的建设是一项庞大的工程，存在大量重复性的工作，这些工作需要领域专家和知识工程师的参与，因此，从零开始为每类专业领域建立本体并不是最适合的方法。最切实际、经济的方法是从现存 Web 中的信息资源中提取语义信息，构建相应的本体。因此，从 XML 文档中抽取隐含语义信息构建描述 XML 文档的形式化语义描述的本体，可以将 XML 描述的信息从语法层提升到语义层。本体可以使用多种语言来进行描述，在这些语言中，OWL 因为其诸多的优点成为 W3C 组织推荐的本体描述语言。

基于以上原因，接下来介绍 XML2OWL 方法，以提取 XML 文档中隐含的语义信息，同时构建与之相应的 OWL 本体，由此实现对 XML 隐式语义

的形式化表达，简化本体构建的难度，拓宽本体构建的途径。

实现框架如下：

图 6-7 给出了 XML2OWL 的架构图。如果 XML 文档没有与之相应的 XML Schema 存在，那么首先通过 Schema 挖掘算法生成一个对应的 Schema。

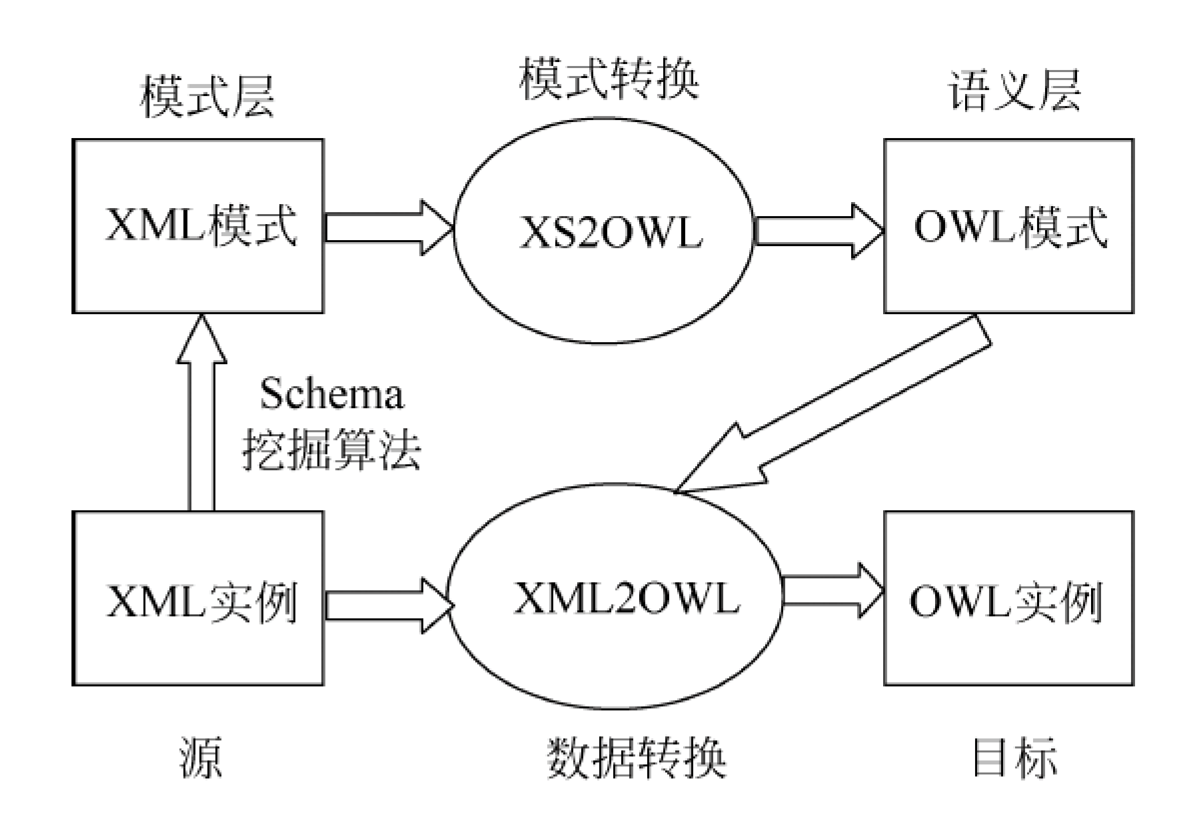


图 6-7　XML2OWL 转换架构

然后以 XML Schema 为输入，将 XML Schema 中的组件映射到一个 OWL 模型，从而获取 XML Schema 中关于 XML 文档的结构、对元素的约束等方面的语义信息。同时，OWL 模型还定义了关于元素之间关系的语义信息。在这个过程中，映射机制还将检查当前元素是否与前一个元素同名，如果两个元素同名，则对当前元素进行重命名。

在 XML Schema 到 OWL 模型的映射完成之后将实现从 XML 实例到 OWL 实例的映射。在这一步系统将对 XML 实例文档和 OWL 本体模型进行处理，在对 XML 实例文档进行遍历的过程中，如果 XML 实例文档中的某个元素与 OWL 本体模型中的某个节点相匹配，XML2OWL 系统将执行 XML 转换指令以生产一个与 XML 数据相应的 OWL 实例。

整个映射的具体过程，XML Schema 到 OWL Model 的转换：表 6-2 给出了 XML Schema 到 OWL 本体模型的映射定义。映射方法将把 XML Schema 中的每一个节点和属性映射到目标本体中的类或对象属性。通过映射所得到的结果是包含了 XML Schema 的结构信息以及其构造器的语义

的 OWL 本体模型。

表 6-2 XS2OWL 转换模型概述

XML Schema 构造器	OWL 表示
ComplexType	Class
Simple Datatype	Datatype Declaration
Element	(Datatype or Object) Property
Attribute	Datatype Property
Sequence	Unnamed Class-Intersection
Choice	Unnamed Class-Union
Annotation	Comment

1. 简单 XML Schema 数据类型

OWL 并不直接支持对简单数据类型的定义,它只允许导入简单数据类型。现有的 XML Schema 的数据类型只有在已声明的情况下才可以应用于 OWL 本体中。所有的简单 XML Schema 数据类型定义在“datatypes” XML Schema 中并为他们生产一个 OWL 数据类型声明。假设 st (name,id,body)是一个 XML Schema 的简单数据类型,其中 body 是 st 定义的主体部分,id(可选)是其标识符,name 是 st 的名字。则 st 将被转化为:①存储在“datatypes”中的简单数据类型 st'(name',id,body);②在主要本体中声明的 dd (about, is_defined_by, label)数据类型。

简单类型 st'与 st 具有相同的 body 和 id,name'的命名规则如下:如果 st 是一个顶层的简单类型,那么 name'与 name 具有相同的值;如果 st 是嵌套在 ae XML Schema 结构中的一个简单数据类型(可能是一个元素,也可能是一个属性),则 name'将根据以下情况来取值:①当 st 中 id 非空时 name'的值为 id 的值;②当 st 中 id 为空时,name'的值为算法 concatenate (ct_name, '_ ', ae_name, '_UNType')的返回值,其中:① concatenate (ct_name, '_ ', ae_name, '_UNType')算法接受任意数量的字符串作为输入并返回他们的连接词;②ct_name 是包含 ae 的复杂类型的名字;③ae_name 是代表 ae 的属性的名字。

数据类型 dd 的声明包含的语义如下：①about 数据类型声明所引用的标识符，其格式是 concatenate（url，name'），其中 url 是“datatypes” XML Schema 的 URL；②is_ defined_by 指定了数据类型定义的位置所在，且其值为 url 的值；③label 是 dd 的标签，其值为 name'的值。

以图 6-8 所示的嵌套简单数据类型为例，复杂类型“a1”中定义的属性“a2”将被转化为图 6-9 中所示的顶层简单数据类型，其 OWL 数据类型声明如图 6-10 所示。

```
<xs:complexType name="a1">
    <xs:simpleContent>
        <xs:extension base="xs:integer">
            <xs:attribute name="a2">
                < xs:simpleType>
                    < xs:restriction base="xs:string">
                < /xs:simpleType>
            < /xs:attribute>
        <}xs:extension>
    </xs:simpleContent>
</xs:complexType>
```

图 6-8　嵌套简单数据类型定义

```
<simpleType name="a1_a2_UNTtpe">
    <restriction base="xs:string:/>
</simpleType>
```

图 6-9　图 6-8 中嵌套类型中表示的顶层简单数据类型

```
<rdfs:Datatype rdf:about="&datatypes;a1_a2_UNType">
        <rdfs:isDefinedby rdf:resource="&datatypes;"/>
        <rdfs:label>a1_a2_UNType</rdfs:label>
</rdfs:Datatype>
```

图 6-10　图 6-8 中简单数据类型的 OWL 声明

2. 属性

XML Schema 属性（attribute）用来描述特征的简单类型。在 OWL 构

造器中用来表示相应特征的元素是数据类型属性(datatype property)。因此,我们将 XML Schema 中的属性转化为 OWL 中的数据类型属性。

假设 a (name, aid, type, annot, ct_name, fixed, default)是一个 XML Schema 属性,其中 name 是 a 的名字,aid 是 a 的标识符,type 是 a 的类型,annot(可选)是 a 的注释元素,ct_name 是在 a 被定义的上下文中复杂 XML Schema 类型 c_type 的名字(当 a 为顶层元素时,ct_name 的值为空),fixed(可选)是 a 的固定值,default(可选)是 a 的默认值。我们将 a (name, aid, type, annot, ct_ name, fixed, default)转化为 OWL 的数据类型属性 dp (id, range, domain, label, comment),其中:①id 是 dp 的唯一的入 rdf:ID,取值 concatenate(name, '-', type)的值;②range 是 dp 的取值范围,其取值为 type 的值;③domain 是 dp 的定义域,取值为 ct _name 的值;④label 是 dp 的标签,其值为 name 的值;⑤comment 是 dp 的文字描述并且以 annot 的值为自身的值。如果 a 的值为空,那么相应的 dp 的值也为空。需要注意的是:如果 a 被指定为一个固定值,则表示在 OWL 类 c 的定义中有一个 c_ type 的值约束。

例如图 6-8 中的"a2"属性将被转化为如图 6-11 中的 OWL 数据类型属性。

```
<owl:DatatypeProperty rdf:ID="a2_a1_a2_UNType>
        <rdfs:domain rdf:resource="#a1"/>
        <rdfs:range rdf:resource="&datatypes;a1_a2_UNType"/>
        <rdfs:label>a2<frdfs:label>
</owl:DatatypeProperty>
```

图 6-11 图 6-8 中"a2"属性在 OWL 中的数据类型属性

3. 元素

将 XML Schema 中用来表示复杂类型的特征的元素转为 OWL 属性:简单类型元素被转化为 OWL 数据类型属性,而复杂类型元素将被转化为 OWL 的对象属性。我们定义 e< name, type, eid, annot, ct_name, substitution_group 为一个 XMLSchema 元素,其中 name 是 e 的名字,eid 是

e 的标识符，type 是 e 的类型，annot 是 e 的注释元素，ct_ name 是在定义 e 的上下文中复杂 XML Schema 类型 c_ type 的名字（如果 e 是一个顶层属性，ct_ name 取值为空），substitution_ group（可选）是被 e 扩展的元素。在 OWL 中将 e 表示为一个（数据类型或对象）属性 p(id，range，domain，label，comment，super_ property)，其中：①id 是 p 的唯一的 rdf：ID，其值取 concatenate (name，'_'，type)的值；②range 表示 p 的值域，其值为 type 的值；③domain 表示 p 的定义域，取值为 cp_name 的值；④label 是 p 的标签，其值为 name 的值；⑤oomment 是 p 的上下文描述，其值为 annot 的值；⑥super_property 是对 p 所指定属性的特化，其值为 substitutio_group 的值。

以图 6-12 中所示的定义在复杂类型 c_t2 条件下的元素 e 为例，e 元素将被转化为如图 6-13 中所示的 OWL 对象属性。

```
<xs:element name="e" type="c-t2"/>
```

图 6-12　嵌套在复杂类型“c-t1”中的元素“e”的定义

```
<owl:ObjectProperty rdf:I D =.. e-c-t2 }..
        <rdfs:domain rdf:resource="#c t1"/>
        <rdfs:range rdf:resource=..#c-t2"/>
        <rdfs:label>e<frdfs:label>
<Iowl:ObjectProperty>
```

图 6-13　图 6-12 中“e”元素所对应的 OWL 对象属性

4. 复杂类型

XML Schema 复杂类型表示具有相同特征的 XML 实例类，恰与 OWL 类被用来表示具有相同属性的个体集相同。因此我们将 XML Schema 中的复杂类型转化为 OWL 类。假设 ct(name，cid，base，annot，attributes，sequences，choices)为一个 XML Schema 复杂类型，其中：①name 表示 ct 的名字；②cid 是 ct 的标识符；③base 表示由 ct 扩展的(简单或复杂)类型；④annot 表示对元素 ct 的批注定义；⑤attributes 表示 ct 的属性序列；⑥sequences 表示 ct 的顺序序列；⑦choices 表示 ct 中的相容元素序列。

如果 ct 是由一个复杂类型派生出来的，则 ct 将被转化为一个相应的 OWL 类 c (id，super_class，label，comment，value_ restrictions，cardinality_ restrictions)，其中：①当 ct 是一个顶层的复杂类型时，id 是一个唯一的 rdf：ID，并以 name 作为它的值。当 ct 是一个与元素 e 复杂的定义嵌套的复杂类型时，name 的值将是一个 concatenate (ct_ name，'_'，element_ name，'_UNType')形式的自动生成的名字，而且该名字是唯一的，在 concatenate (ct_ name，'_'，element _name，'_UNType')中，ct _name 是包含元素 e 的复杂类型的名字，element_ name 是元素 e 的名字，如果 e 是一个顶层元素，那么 ct_name 取'NS'的值；②super _class 给出了哪些类是通过 ct 派生出来的，并以 base 为值；③label 表示 ct 的标签，取 name 的值；④comment 表示 ct 的上下文描述，取值为 annot 的值；⑤value_restrictions 是指对 c 的属性的值约束集；⑥cardinality_ restrictions 是指赋予表示 ct 属性和 ct 的 sequence/choice 元素的基数约束集。

当 ct 是由简单类型派生出来的时候，我们将 ct 转化为 OWL 类 c(id，label，comment，value_restriction，cardinality_restrictions)，其相应的选项与从复杂类型派生出来的复杂类型的类中具有相同的语义。简单类型的派生通过基数 1 的数据类型属性 ep(eid，erange，edomain)来表示，其中：①eid 表示 ep 的唯一的 rd：ID，取值为 concatenate (base、'_content')的值；②range 表示 ep 的定义域，以 base 为它的值；③domain 表示 ep 的值域，其值为 c 的 id 的值。

ct 中定义或引用的元素和属性将被转化为相应的 OWL_ DL 构造器。

我们以图 6-8 中所示的复杂类型 al 为例。复杂类型 al 将被表示成图 6-14 中所示的带有 content_xs_integer 数据类型属性的 ctOWL 类，其 content_xs_integer 表示 al 是 xs：integer 的派生。

5. 特定顺序和选择顺序

XML Schema 的特定顺序(sequences)和选择顺序(choices)对 XML 元素容器是必不可少的，被用来定义复杂类型和模式群的的语境。特定顺序

```
<owl:Class rdf:ID="a1">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onPropertg rdf:resource="#a2_a1_a2_UNType"/>
       <owl:maxCardinality rdf:datatype="&xsd;integer">
       </owl:maxCardinality>
      </owl: restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
       <owl:onPropertg rdf:resource="content_xs_integer"/>
        <owl:cardinality rdf:datatype="&xsd;integer">
       </ow1:cardinality>
     </owl:Restriction9>
    </rdfs:subClassOf>
    <rdfs:label>a1</rdfs:label>
</owl:Class>
<owl:DatatypePropertg rdf:ID="content_xs_integer"/>
    <rdfs:domain rdf:resource="#a1"/>
  <rdfs:range rdf:resource="&xs;integer"/>
</owl:DatatypeProperty}
```

图 6-14 图 6.8 中复杂类型“al”所对应的 OWL 类

和选择顺序之间的主要区别是特定顺序中的元素是有序的，而选择顺序下的元素是无序的。我们将特定顺序和选择顺序都转化为未命名的 OWL_DL 类，该未命名的 OWL_DL 类以又 sequence/choice 选项(元素，特定顺序，选择顺序)进行复杂基数约束为特征，该未命名的 OWL_DL 类将被定义在 sequences/choices 被定义或引用的复杂类型所对应口 OWL 类定义中。

构造器的下界最小基数约束表示该 sequence/choice 选项有一个 i_min_occurs×s_ min_ occurs 的值，构造器的上界最大基数约束表示该 sequence/choice 选项有一个 i_max_occurs×s_ max_occurs 的值，其中：①i_min_occurs 是该项的 minOccurs 的值；②s_min_occurs 是特定顺序 sequence 的 minOccurs 属性的值；③i_max_occurs 是该项的 maxOccurs 属性的值；④s_max_occurs 是特定顺序 sequence 的 maxOccurs 属性的值。此外，sequence/choice 选项的基数必须在[i_min_occurs—i_max_occurs]的范围内。

特定顺序 sequence 的选项必须按照特定的顺序出现。因此，特性顺序 sequences 被转化为未命名的类以形成对他们的选项的基数约束的交集。需要注意的是：当一个特定顺序 sequence 选项被包含在一个以无穷大为上界的特定顺序 sequence 中，该选项没有最大基数约束时，特定顺序 sequences 基数不可被计算。此外，特定顺序 sequence 的元素排序的相关信息不能再 OWL 中表示出来。

以图 6-15 中所示特定顺序为例，该特定顺序 sequence 被定义在复杂类型 c_ t1 中。则该特定顺序将被表示成图 6-16 中所示的 c_t1 类中的未命名类。

```
<xs:sequence minOccurs="2" maxOccurs="2" >
     <xs:element name="e1" type="xs:string />
     <xs:element name="e2" type="xs:strut g" maxOccurs="3"/>
<xs:sequence>
```

图 6-15 复杂类型 c_typel 环境下的特定顺序 sequence 的定义

```
<owl:Class>
  <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:onProperty rdf:resource="#e1_xs_string"/>
         <owl:cardinality rdf:datatype="&xsd;integer">
         </owl:cardinality>
      < owl:Restriction>
      <owl:Restriction>
       <owl:onProperty rdf:resource="#e2_xs_string"/>
      <owl:minCardinalityrdf:datatype="&xsd;integer">
      </owl:minCardinality>
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#e2_xs_string"/>
      < owl:maxCardinality rdf:datatype="&xsd;integer">
      </owl:maxCardinality>
      </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

图 6-16 图 6-15 中的特定顺序在 OWL 中的顺序

选择顺序 choice 的选项可以以任意顺序出现。因此，我们将选择顺序 choice 转化为未命名的类以形成对选择顺序 choice 的元素的基数约束的允许组合的并集。需要注意的是：当选择顺序 choice 中的某一个 choice 选项的出现上限为无穷大的时候该选择顺序基数不可以被用来计算。

表示 XML Schema 中的选择顺序 choices 和特定顺序 sequences 的未命名的类是通过上述的算法来实现的。需要注意的是：如果 sequence/choice 发生的最大数一个很大的数(不是无穷大)时，手工生产约束是繁琐而且很耗时间的，甚至会产生很多错误，因此在实际操作中将不可能实现。

6. 引用

被复杂类型定义所引用的 XML Schema 属性，属性组，元素和模型组将被转化为 OWL_DL 数据类型(如果他们本身是属性或者简单类型元素或者他们包含属性或简单类型元素)或对象(当他们包含复杂类型元素)属性。假设 ref (ae)是一个引用类型，ref (ae)在一个复杂类型 ct 中引用了 XML 的 ae 属性或元素。则该引用将被表示为(数据类型或对象)属性 rp (id，domain)，其中 id 表示 rp 的 rdf ：ID，其取值为表示 ae 的属性的 rdf：ID 的值，domain 表示 rp 的定义域，其取值为表示 ct 的 OWL 类 c 的 rdf：ID 的值。

7. XML 实例文档到 OWL 实例文档的映射

从 XML 实例文档到 OWL 实例文档的映射过程的输入包括前一步骤中生成的 OWL 本体模型和给定的 XML 实例文档。映射的第一步是为 OWL 实例文档指定命名空间。我们使用 http://www. w3. org/2002/07/owl 为支持 OWL 语义的命名空间，同时我们使用 http://www. w3. org/1999/02/22-rdf-syntaxns 名称空间来为 RDF 提供语义支持。然后将确保 OWL 实例文档中的每一个类和属性的名字的唯一性，由于对 XML 实例文档的处理时从根节点开始的，将默认为每一个元素添加一个 ID 以区分同名元素，当遇到同名元素时我们将前一个与当前元素同名的元素的 ID 值加 1 赋给当前元素。

6.4 基于 Ontology 的领域知识库构建

知识服务作为面向内容的增值服务，需要对知识进行系统化、综合化、深入化地加工、组织和处理。知识库和知识发现是知识服务的关键技术，特别在基于知识库的知识发现领域，知识库必须是共享的学习型的，将本体技术引入此领域根本上解决了知识库机器学习的瓶颈问题。

6.4.1 领域知识

知识库是为解决特定的问题，采用合理的知识表示和组织方式对该问题相关概念及其关系存储和管理的知识集合。知识库包括两个部分：一部分由相关术语及其关系构成，另一部分由术语的具体实例及其关系的实例构成。要研究领域知识库首先要理解领域知识的概念。领域知识是一个源于人工智能领域的术语，在人工智能领域，领域知识主要应用在基于知识的专家系统和自然语言理解的系统中。领域知识是指在某一领域内的概念、概念之间的相互关系以及有关概念的约束的集合。

根据不同领域和不同的应用需要，领域知识这一术语的定义也有所不同。知识工程对领域知识进行了三个方面的描述：

(1) 领域知识是一个概念模型，这个概念模型包含概念和概念之间的关系；

(2) 领域知识是概念和概念之间的约束；

(3) 领域知识是陈述如何推导或计算出新概念和新概念之间关系的规则。

在数据挖掘处理的过程中，领域知识是指一个专门领域的重要问题或概念以及这些问题和概念之间的相互关系。领域知识库是经过合理组织的关于某一特定领域的陈述性知识和过程性知识的集合，知识通过一定的表示，存储在知识库中。在数据库知识发现系统的应用及其他相似系统的应用中，领域知识定义为没有在数据库中明确表达的知识。

6.4.2　领域知识库和本体

从知识表示角度考虑，本体和知识库有类似之处，都是对一个具体或是抽象领域中包含的知识的定义、表示和组织。根据 Gomez 定义：知识库是知识系统的知识模块，它包含特定领域抽象或特定知识，这些知识以机器可读的格式表达。知识库的知识可以是描述性或过程性。Gruber 认为本体是概念层次上对概念化的清楚描述，注重概念层次上术语及术语间关系的表述。可见，本体侧重于领域知识内容的描述，而知识库更侧重于领域知识的表示、组织和存储。

在本体的概念提出来之后，理想的领域知识库是应该建立在领域本体的基础之上的，根据领域本体中的全部或部分概念生成系统所需的知识库。而且本体为人们描述目标世界提供了一组通用词汇，而这种通用的词汇正是实现知识系统化的基础。通用词汇和知识的系统化有利于实现知识的标准化。因此，知识库应该建立在本体的基础之上，本体是知识库建立的基础。

6.4.3　基于本体构建领域知识库的优势

使用本体建模的方法和相关知识表示的标准及交换协议建立知识库，便于研究者之间的交流、协作开发，对计算机系统来说可以实现不同领域，不同模型之间的跨平台的方法、数据、任务、工具的转换和共享。使用本体建模的优点有很多，主要有：

（1）可重用：Ontology 是某领域的基本概念、属性、处理方法和内在关系的形式化表述，这种表述可以被重用和共享；

（2）便于查找：便于基于 Ontology 的具有一定智能的以内容为目标的查找；

（3）可靠性：模型的形式化表述便于正确性检查；

（4）有助于任务解析：用 Ontology 进行规范任务的形式化表述，有助于任务的解析；

（5）可维护性：使用 Ontology 进行系统建模，使系统结构、文档和编码都更加清晰，使系统的维护更加容易。

利用 Ontology 的思想对领域知识库进行建模，可以使相互独立的层次有机地组成一个完整的系统，可以实现领域知识的共享和重用，而且领域知识概念模型的形式化描述便于正确性检查，可以使领域知识库的结构更加清晰，更有利于领域知识库的维护。

领域知识库是一个在文本的主题和内容分析研究中，用于揭示领域特征概念与领域特征概念之间以及领域特征概念所具有的属性之间的关系的常识性知识库。领域知识库面向计算机，是借助于计算机建立的。领域知识库反映了领域特征概念之间和领域特征属性之间的各种关系，领域知识库着重体现的是领域特征概念的上下位关系。

6.4.4 领域知识库的构建

构建领域知识库，首先要提取领域特征属性，应遵循以下三个基本的要求：

（1）领域特征属性能够描述某一领域的领域特征概念，且不易于再分割；

（2）领域特征属性一定要能够描述某一领域中全部的领域特征概念；

（3）领域特征属性是稳定的，是必须确定的。

1. 知识表示

知识表示（Knowledge Representation）即知识表达或知识描述，是为描述世界所做的一组约定，是知识的符号化过程。各种不同的知识表示方法是各种不同的形式化的知识模型。可以用不同的方式来描述同一事物，那么对于同一表示模式的知识，也可以采用不同的表示方法。

知识表示方法是知识工程师对领域知识的事实和关系的一种模型化。知识的表示方法有很多。如：一阶谓词逻辑表示法、产生式表示法、语义网络表示法、框架表示法、脚本表示法、状态空间表示法，等等。

1）一阶谓词逻辑表示法(First order predicate logic)是最早使用的知识表示方法，它具有简单、自然、精确、灵活、模块性好等优点，它的推理机制采用归结原理，这种推理方法严格、完备、通用，比较适用于用定理方法求解问题的系统。一阶谓词逻辑表示法的缺点是难以表达和加入启发性知识及元知识，不易实现非单调和不精确推理。

2）产生式(Production)是在 1943 年由逻辑学家 Post 提出的，称 Post 演算。但是由现在的知识表示的研究发展形成的产生式与 Post 早期提出的概念大不相同。产生式表示法又称规则表示法。产生式规则的一般形式为：if＜前提＞then＜动作＞(或＜结论＞)，它的推理机制以演绎推理为基础，推理系统也称为产生式系统。产生式系统具有自然、灵活、模块性好、通用性强等优点。它是目前应用最广泛的知识表示方法。产生式表示法的缺点是效率低，表达能力低。

3）语义网络最初是用来描述英语的词义，最早是作为知识表示工具讨论的，语义网络的实质是基于广义图的表示法，一由结点和弧(包括超弧)所组成，其中结点代表概念，结点之间的弧或超弧说明了它们之间的相互关系。用语义网络表示知识能够直接而明确地表达概念之间的语义关系，相关事实可以从其直接相连的结点中推导出来，而不必遍历整个庞大的数据库，重要的相关性能被明确而清晰地表达出来，着重于表达语义关系知识，体现了联想思维过程，易于对继承层次进行演绎。语义网络表示法的缺点是不能像逻辑方法那样保证网络操作所得推论的严格性和有效性，不便于表达判断性知识和深层知识。通常将语义网络和其他知识表示方法结合起来运用。

4）框架(Frame)是 1975 年由著名人工智能学者 Minsky 在研究计算机视觉问题时提出的，知识框架表示法的基本出发点是：把人们头脑中的不同概念看成是一定的知识体或一定的数据结构，即看作框架。所谓框架就是一种描述某种形态的数据结构。框架常用于描述具有固定形式的对象。

框架表示法的优点很多：

(1) 框架的嵌套式结构可以由浅入深地对事物细节进行知识表达，框架

的继承性可实现高效的推理；

(2) 槽的过程附件不仅提供了附加推理机制，还可以进行矛盾检测，框架表示有利于“领域知识库”的一致性维护；

(3) 框架常用于描述给定事物所具有的一组属性，是更加面向语义的知识表示方法。

5) 脚本表示法是由 Roger C. Shank 提出的，它的结构类似于框架，用于描述固定的事件序列。与脚本相比，框架是一种通用的结构，而脚本的形式比框架的形式应用范围窄，它对于表示某些专门知识更为有效，如理解故事情节等。

6) 状态空间表示法是知识处理学中最基本的形式化方法，是其他形式化方法和问题求解技术的出发点。问题的状态空间表示问题的全不可能状态及其相互关系，可以用一个赋值有向图来表示。问题的状态空间常记为三元组(S,O,G)，其中 S 为问题的所有初始状态的集合，O 为所有操作的集合，G 为目标状态的集合。在状态空间表示法中，问题求解的过程转化为在图中寻找从初始状态 S 出发到目标状态 G 的路径问题。

7) Petri 网是由 C. A. Petri 博士在 1962 年提出的。利用 Petri 网表示知识，根据 Petri 网的特性，我们就能处理并行推理、无回溯推理、反向推理等问题。利用 Petri 网可以模拟逻辑运算、产生式规则、语义网络、框架、状态空间等多种功能。因此 Petri 网可作为一种通用的知识表示形式。

除了以上的几种主要知识表示方法以外，还有许多知识表示方法。如关系表示法(又称“特征表”表示法)、“与/或”图表示法、概念从属表示法、直接表示法、过程表示法、面向对象的表示法，等等。

领域知识库对领域特征概念的描述也要有一套规范的方法和规定来保证最大限度地表现领域特征概念之间以及领域特征概念属性之间的相互关系。在解决某一问题时，不同的表示方法可能产生完全不同的效果。为了有效的解决问题，必须要选择一种良好的知识表示方法。选择合适的知识表示方法通常从四个方面要求：

(1) 表示能力。在语义上能够较好地反映领域专家的知识的含义，在语

法上能够方便识别和处理；

(2) 推理效率。使推理机寻找答案的搜索路径尽可能短，并且不出现循环；

(3) 易于维护。对知识进行添加、修改、删除、一致性维护等操作易于实现；

(4) 正确性。知识表示在语义上能准确地表达领域专家知识的含义，这是推理正确性的基础。

在领域知识库中，每个框架表示领域知识库中的一个领域特征概念。框架中的槽描述领域特征概念的各种属性。其中，“领域特征属性”是我们重点要研究的内容。

基于 OWL 语言的本体是描述某一领域共享“领域特征概念”模型的明确的形式化规范说明，因而本文讨论基于本体的知识库构建。

2. 知识库构建流程

领域本体知识库的构建一般包括三部分：知识获取、构建本体概念树和知识的表示，前两部分起着至关重要的作用。知识获取是将从该领域采集的数据转换成容易存储容易处理的形式，使计算机可以读取。概念树的组织过程主要是把从该领域抽取的概念以及其之间的关系以树形结构表示出来，这两部分的处理关系到之后本体推理中信息检索的质量问题，因为领域知识的限制，这两部分的完成需要相关专家的参与。知识表示是实现本体结构与信息数据的连接。

该建模过程有如下优势：①因为本体结构与信息数据的精确映射，可以准确地表示领域知识；②领域专家的参与可以确保定义概念关系的正确性，实现推理的一致性检测，提高检索的精确性；③从描述性和验证性两方面对所建本体进行评估，不断修正和完善本体结构。

对知识库中知识的获取和概念树的组织过程，知识库的构建流程如图 6-17 所示。

这里以中医喘证知识本体[6]为例进行说明。根据以上理论，通过人工

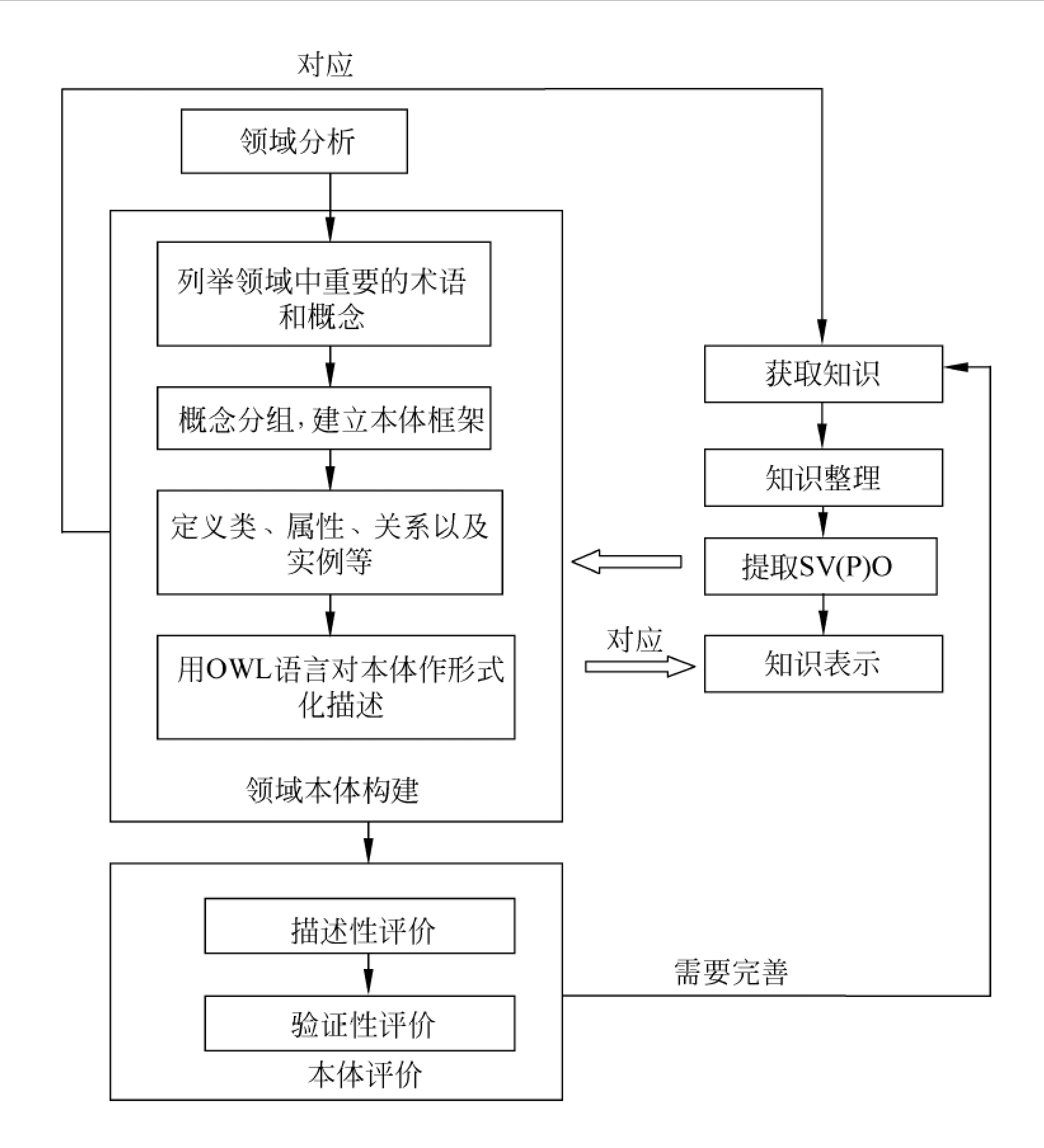


图 6-17 知识库的构建流程

收集和整理《丁甘人医案 · 伤寒案》、《中华医典》中所收录的名医案中涉及到的喘证案例，选取 600 余例喘证医案为研究对象，在领域专家的参与下并参考中医学相关标准，在排除概念的冗余性、歧义性及保证概念术语正确的前提下，选择"喘主证"、"喘息"、"喘逆"、"喘鸣"、"咳喘"及"上气"等关键词作为研究核心概念，筛选整理医案，最终确定相关属性字段千余个，建立较为完善的中医喘证医案数据库，如图 6-18 所示，将此作为研究对象的形式背景。

（1）定义喘证本体结构的类及其层次关系

在中医专家的参与下对抽取的重要概念自顶至下定义出基本的类和层次关系，并用建模工具 Protégé4. 1. 0 建立中医喘证领域本体类关系的初始模型。其中，定义的类与本体结构中的类一致，对象与本体结构中的实例一致，例如，外感型喘是实喘的子类，而实喘又是喘证的子类，用本体语言描述

	A	B	C	D	E	F	G
1	编号	喘主症	喘轻重	**痰象**	伴随症状（寒热、汗出、口渴、	舌象	脉象
2	目标	咳喘	中	痰白滑	纳呆，便秘	苔白腻	滑
3	1	喘	重	痰黄	状热	苔黄腻	滑数
4	2	咳喘	中	痰白滑		苔白腻	滑
5	3	喘息	重	痰白滑	自汗	舌淡，苔薄白	细软，滑
6	4	气短、喘	中		小便短赤，便秘	苔白	沉实
7	5	气短、喘	轻		小便少，泄泻	舌红，苔白腻	沉弱
8	6	喘促	中		微热	舌淡，苔白微腻	浮数
9	7	喘促	中		状热	舌淡苔腻微灰	沉数
10	8	微喘	中		状热，口渴	舌质红，苔微黄	浮数
11	9	喘促	重	痰黄	状热，泄泻，纳呆，便秘	舌红，无苔	虚
12	66	喘促	重	痰黄	微汗，纳呆，口渴，泄泻，小便少		
13	67	喘促	重	痰黄	状热	舌红少津，苔薄白而	沉数浮大，指
14	68	喘挣	中		壮热，纳呆		
15	69	喘			壮热，口渴，便秘，小便短赤，	苔黄燥	沉数，指纹粗
16	71	喘	重		壮热，口渴，小便短赤，纳呆	舌红，苔燥	浮弦，数
17	72	喘	重	痰清稀	纳呆，口渴，小便清长	苔白滑而腻	弦滑，弱，细
18	75	喘咳	中	痰清稀	纳呆，便秘，小便短赤，口渴	苔白腻	沉迟
19	76	喘咳	中	痰清稀	纳呆，小便清长	舌青色，苔白滑厚腻	弦滑
20	78	气短	中	痰少而黏	潮热，恶寒，自汗，盗汗，	苔白腻	虚数
21	81	喘挣	重		壮热	苔白滑	紧，指纹青黑
22	84	喘满短气	中	痰白	自汗，恶寒，口渴		

图 6-18　喘症医案数据库

这种关系如下：

```
<owl: Class rdf:ID= "外感型喘">
<rdfs:subClassOf>
<owl:Class rdf:ID= "实喘"/>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about = "实喘">
<rdfs:subClassOf>
<owl:Class rdf:about = "喘证"/>
</rdfs:subClassOf >
</owl:Class >
```

（2）定义并应用各类之间的关系

定义各类之间的关系，每一种关系都是概念与概念间的一种映射，可以看成是二元组或多元组函数，函数定义域和值域的取值就是喘证领域内定义的类和子类的对象，每种关系相对应的 ObjectProperty 属性的 domain 子属性可以用来设置函数的定义域，关系对应的 ObjectProperty 属性的 rang 子属性可以用来设置函数的值域，如此可以使本体体系中各类的关联更加简单，以下显示的是 ObjectProperty 属性“痰象”以及它的特殊性质，其中

“痰象”的定义域是案例，值域是由“痰白”、“痰黄”和“痰清稀”组成的并集构成，可以看出值域的并集构成了值域，而且值域还有传递属性。

```
(<owl: TransitiveProperty>)
<owl:TransitiveProperty rdf:about=" #痰象 "
<rdfs: domain rdf: resource="#案例 "/>
<owl: inverseOf rdf: resource="#构成 "/>
<rdfs: range>
  <owl: Class>
    <owl: unionOf rdf: parseType="Collection">
      <owl: Class rdf: about="#痰白 "/>
      <owl: Class rdf: about="#痰黄 "/>
      <owl: Class rdf: about="#痰清稀 "/>
    </owl: unionOf>
  </owl: Class>
</rdfs: range>
<rdf: type rdf: resource=http://www.w3.org/2002/07/owl#ObjectProperty/>
</owl: TransitiveProperty>
```

(3) 确定各本体类的属性及其属性间的关系

确定本体类的属性时要给出属性名、属性值及属性类型等，这些内容对应本体系统中的数据类型属性。在比较属性之间的关系时可以利用属性的这些内容，也就是说个体之间通过属性连接起来。

本体结构中概念间的关系有以下几种：

概念之间部分与整体的关系(Part of)，例如，键盘和电脑，因为概念键盘是概念电脑的一部分。

概念之间的父子继承关系(Kind of)，例如，汽车和乘用车，乘用车是汽车的父概念，汽车是乘用车的子概念。

概念的实例和概念之间的关系(instance of)，例如，商用车和客车，客车这一概念是商用车这一概念的一个实例。

某概念是另一概念的属性(attribute of)，例如，成绩和学生，概念成绩是学生这一概念的一个属性。

概念之间的联系使概念间有多种多样的语义关系，利用规则推理就可以得到其蕴含的关系。

（4）定义概念的词义扩展关系

比如相同、相异、相反及父子继承关系等；只有本体模型能够对词语间的相同、相异及相反及父子继承等关系进行查询，才能达到对基于关键字的语义检索。例如，“喘息”和“微喘”意思相近，如果把其中的一个词语作为核心词汇输入系统，既可以得到核心词汇本身的信息，还能得到与核心词汇意思相近的信息内容。这也是将来提升案例检索系统查全率的关键技术。

（5）实例填充

将具体的例子写入(1)中定义的类，同时对例子的属性及其属性值进行定义和扩充。

（6）修正本体模型

利用 Jean 推理机[7-8]完成对初始构造的本体结构的推理，检验该本体结构模型是否合理及本体结构中的概念是否一致，修正并完善不符合一致性的概念。

（7）确定本体的存储方式

在对本体进行存储时可以将其以 OWL 或 RDF 等文件格式存储，也可以用数据库的形式进行存储。在本文的研究中，我们选择 OWL 的文件格式存储喘证本体模型，采用关系数据库存储知识库中的数据信息。

应用 Protégé 工具建立的中医喘证本体片段如图 6-19 所示。

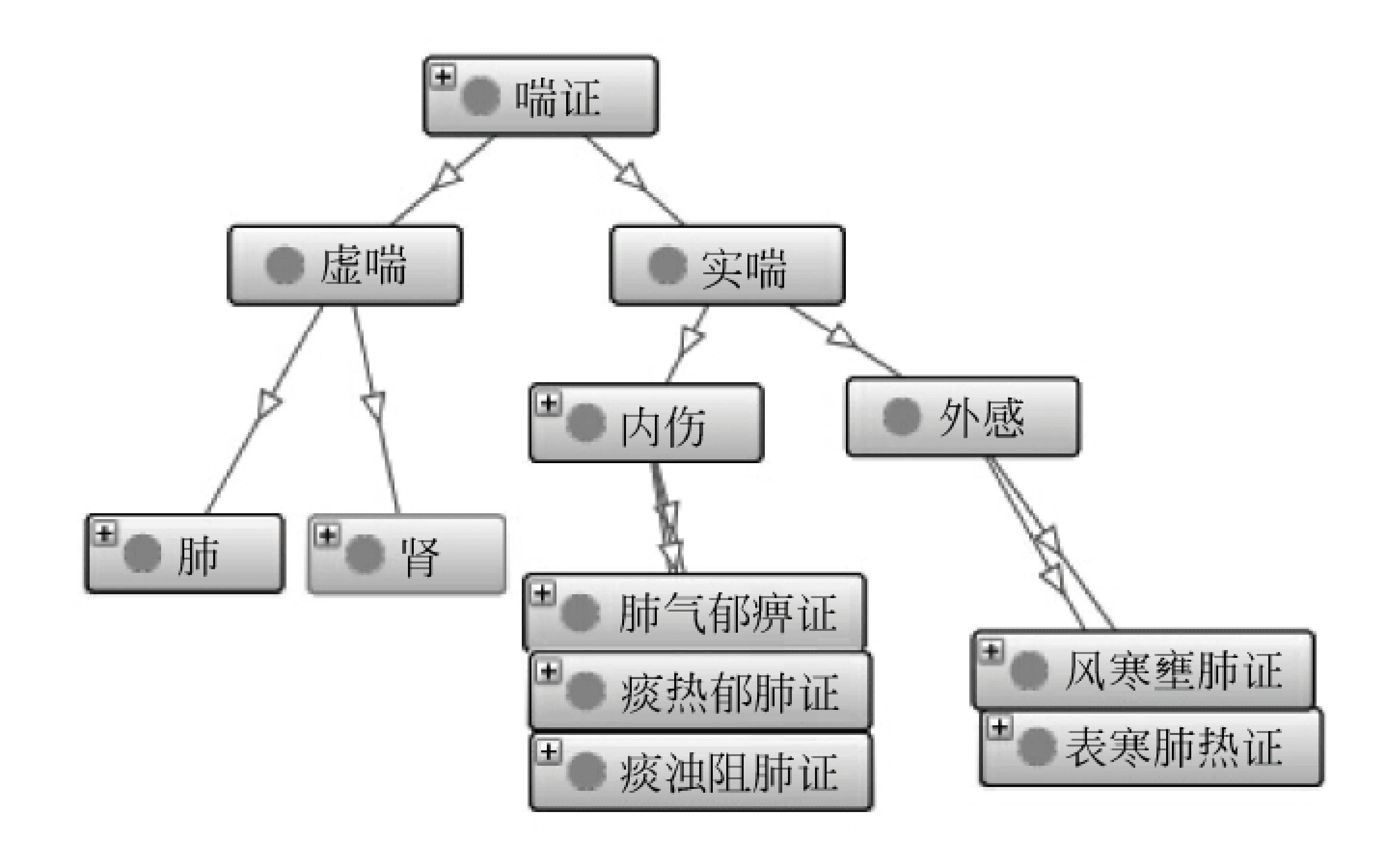


图 6-19　中医喘证本体片段

参考文献

[1] 王益广，熊松韫，李险峰等．知识的描述与发现[J]．图书馆理论与实践，2007.4：55-56

[2] 何琳．领域本体的半自动构建及检索研究[M]．南京：东南大学出版社，2009.12

[3] 李恒杰，李军权，李明．领域本体建模方法研究[J]．计算机工程与设计．2008，29(2)：381-384

[4] 马晓丹，邓晓楠，彭文娟等．基于领域本体的知识库架构和实现[J]．河北联合大学学报，2012(10)：42-47

[5] 谭介平．XML 数据到 OWL 本体的转换方法的研究[D]．华东交通大学硕士学位论文，2010

[6] 阎红灿，王会芳，马会霞．中医喘证医案的本体表示及知识库构建．医学信息[J]，2015，28(1)：1-2

[7] YAN Hongcan，LIU Chen．Knowledge reasoning and Tableau Algorithm improving based on Rough Description Logics FSKD2012，2012

[8] 冯志勇，李文杰，李晓红．本体论工程及其应用[M]．北京：清华大学出版社，2007

第 7 章　万维网信息资源的描述与发布

XML 是结构化标记语言，即实现“文档结构化”的语言规范，是与特定领域有关的、具有语义和结构化等特点的元标记语言。XML 将文件的内容和外观进行分离，其所具有的可延伸性及自我描述特性，使得 Web 文件可以在全球信息网或企业间的应用程序中自动传输、处理及储存，不同厂商的电子商品目录可以共享，信息的搜索变得更为准确快速，不同系统间的信息流通更加顺畅，给 Web 应用乃至网络计算注入了新的活力。

7.1　XML 的有关技术规范

自从 XML1.0 规范发布之后，XML 的有关技术规范不断涌现。W3C 在 1999 年先后推出了 Namespaces in XML（XML 中的名字空间）、CSS2、Associating Style Sheets with XML Documents（将样式表关联到 XML 文档）等推荐标准，与 XML 有关的重要技术规范还包括 DOM、XSL、XLink 和 XML Schema 等[1]。2001 年 12 月 13 日，XML 1.1 作为一份工作草案被发布，并作为一项候选推荐发布于 2002 年 10 月 15 日，XML 1.1 允许在名称中使用几乎所有的 Unicode 字符。

7.1.1　DOM

围绕 XML 出现的各种标准的应用编程接口（API），对 XML 应用开发来说无疑是十分重要的。应用开发者可以使用这些标准的接口来获得和设置 XML 文档中的元素、属性、数据内容等。在这些 XML 的应用编程接口中，最重要的是 W3C 制定的 DOM[2]（Document Object Model 文档对象模

型)。DOM 是基于文档的树状结构,它提供了用来表示 XML 文档和 HTML 文档的一组标准的对象组合,这些对象的标准模型以及存取和操纵它们的一个标准接口。DOM level 1 已于 1998 年 10 月推出,它是 W3C 为 XML 文档和 HTML 文档制定的一个独立于平台和语言的编程接口标准,使得程序和脚本能以标准的方式存取与更新文档的内容、结构和样式。其中 DOM level 1 的核心(Core)部分提供了能表示结构化文档的一组低层的基本接口集,并定义了用来表示 XML 文档的扩展接口。DOM level 1 的 HTML 部分在上述基本接口集的基础上定义适用于 HTML 文档的额外的高层接口,DOM level 2 尚在制定过程中,它建立在 DOM level 1 之上,给出了用来创建和操纵文档结构和内容的一组核心接口,并提供一组可选模块,这些模块包含了为 XML、HTML、抽象视图、类属样式表、层叠样式表、文档结构的遍历、Range 对象等特制的专用接口。

7.1.2 XSL

XML 的一个最重要的特性是能够把内容和显示格式分开,这个特性给用户带来了很大的好处,可以让不同的用户按照各自希望的格式显示同一 XML 文档的数据内容。这也就意味着 XML 文档本身并没有关于格式方面的信息,为 XML 文档提供格式信息的是样式表,适用于 XML 文档的样式表语言有 XSL 和 CSS2 语言。CSS2 语言既可以用于 HTML 文档也可以用于 XML 文档;而 XSL 是专为 XML 设计的样式表语言,并采用 XML 语法,目前处于 W3C 的工作草案阶段。XSL 的优势在于它可以用于转换。当然 XSL 也可以把 XML 文档转换为 HTML 格式,而且同一个样式表可以用于多个具有相似源树结构的文档。显示的媒介不仅限于 Web 浏览器,还可以是印在纸上的书和报告等。

处理 XSL 样式表的是 XSL 样式表处理器,样式表处理器接受一个 XML 文档或数据,以及 XSL 样式表,输出特定样式的显示,其显示格式根据 XSL 样式表确定。这个处理过程分两步进行,首先从 XML 源树构建一棵结果树,然后翻译结果树,产生作用于显示器或纸或其他媒介的显示。第一步

被称为树转换，在 XSLT（XSL Transformation）中详述；第二步称为格式化，在 XSL 规范中有详细说明。

7.1.3　Xlink 简介

XLink 是一种用 XML 元素向 XML 文档中加入链接的机制。它提供了比 HTML 更加灵活的链接机制，不仅支持 HTML 的单向链接，还支持多目的、多方向链接，它甚至还允许链接单独提出来存放在数据库中，或者是单独的文档中。XLink 通过 URI 引用来定位资源，URI 引用由 URI 和一个可选的块标识符构成，两者用井号 # 分开。对于定位 XML 文档来说，块标识符使用 Xpointer 规定的格式。对于链接元素来说，应用软件可以通过获取元素类型及属性名字或通过名为 XLink 的名字空间来辨认和处理链接。这两种方法都能很简单地确定链接元素。

链接可以分为简单链接（Simple Links）和扩展链接（Extended Links）。简单链接功能相当于 HTML 中的＜A＞标记，但与 HTML 不同的是，在 XML 中没有规定必须使用＜A＞之类的元素进行链接。这一点很重要，因为这就允许用户根据需要在同一文档中加入不同的具有自己独有属性的链接元素，充分体现了 XML 的灵活性和可扩展性。（1）扩展链接是 XLink 为支持多方向多目的而提出的，和简单链接不同，它可以有多个目标。（2）多方向（multidirectional）链接是指对链接的操作可以从任一个链接资源开始。为支持多方向链接，XLink 允许链接元素本身处于链接资源之外。链接资源可以是 XML 文档中的任何元素，也可以是整个文档。由于链接元素本身处于链接资源之外，因此这种链接可以自由组织多个文档之间的关系。当然 HTML 也可以通过在所有文档中加＜A＞来实现同样的功能，但若要对这些关系进行修改，HTML 需要修改所有文档，而外部链接只要修改一个文档中的一个链接元素就可以，显然要比 HTML 方便得多。这也是 XLink 链接机制的优点所在。

扩展链接组是一种特殊的扩展链接，它定义了一批包含扩展链接和其他扩展链接组的扩展链接文档，要求处理软件能够同时处理这批扩展链接

文档，而不是一个一个的来处理。对浏览器来说，要求最好能够把这些链接的内容同时显示。扩展链接组是一个全新的概念，它要求应用软件在处理时能够理解和调用组中成员之间的关系。它的实现仍然是个重点和难点，对这个问题 W3C 组织也还在讨论中。

7.1.4 XML Schema

我们知道 DTD 缺乏对 XML 文档的内容及其语义的约束机制，这将限制 XML 处理器进行有效的类型检验，应用软件开发者将不得不专门编写有关类型检验的代码。因此有必要为 XML 建立一个更全面的有效性约束机制，使 XML 处理器更好地进行有效性检验，这样就产生了 XML Schema Language。

用 XML Schema Language 书写的 Schema 文档定义了相应 XML 文档的规则，以约束其数据元素及其关系[3]。首先 Schema 文档从数据结构和数据类型两方面更严格地约束相应 XML 文档，它可以定义 DTD 所无法定义的规则，而 DTD 仅从结构上对 XML 文档进行有限的约束。其次，DTD 语言有其独立的语法形式，而 XML Schema Language 实际上是 XML 语言的一个应用(类似 HTML 与 SGML 语言的关系)，因此 Schema 文档本身就是一个 XML 文档，可以用 XML 工具进行分析，这样 Schema 文档也就可以用现有的 DTD 语言加以描述。它们之间的关系如图 7-1 所示。

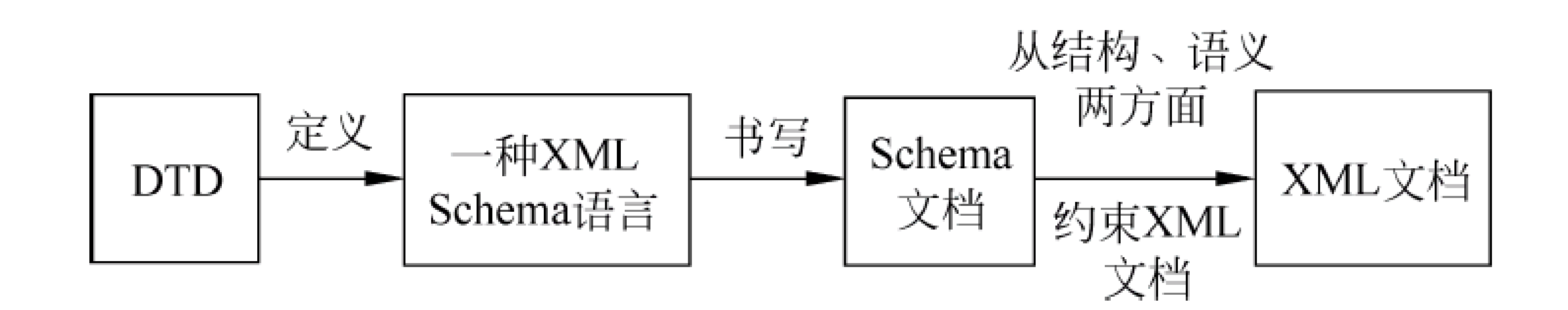


图 7-1 Schema 文档和 XML 文档放入关系

7.2 建立 XML 应用过程

虽然人们对 XML 的某些技术标准尚有争议，但是人们已普遍认识到 XML 的作用及用途，XML 文档无疑将成为 Web 资源的重要组成部分，而且

基于 XML 的文档资源使 Web 搜索引擎的智能化变得容易起来。XML 能够用来建立 HTML 所不能达到的多层 Web 应用,特别是 XML 在集成异构数据源、本地计算、数据的多种显示、Web 应用的互操作和集成等方面有重要应用。另外,数据交换是 XML 的最重要的用途之一,XML 使不同计算机应用系统之间交换数据变得容易起来。这是因为它的可扩展特性和文档中的元数据,特别是 XML 在电子数据交换、Agent 软件设计、元素的交换等方面有重要应用。

建立 XML 应用的步骤一般包括以下 4 个部分,如图 7-2 所示。

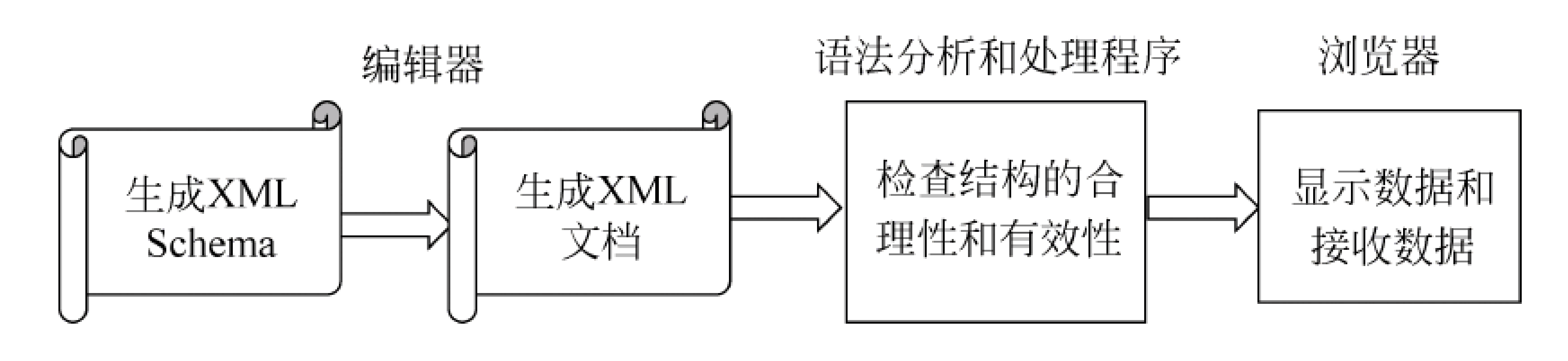


图 7-2　建立 XML 应用过程

1. 构造 XML 标记语言

XML 允许任何人创建他自己的标记语言,但是设计一个好的标记语言并非易事。首先通过 DTD 来设计标记,其次用自然语言解释这些标记的含义,以便程序员能正确处理 XML 文档中的元素。然而,在很多场合用 XML 1.0 中规定的 DTD 来定义标记显得有些不够,另外人们也希望用 XML 文档来定义标记结构及约束,为此要设计一个 XML Schema Language,以便用 XML 文档来定义增强的 DTD。由 Microsoft 及其合作伙伴提交的 XML-Data 正是这种语言的一个典型代表。

2. 生成 XML 文档

XML 文档是纯文本文件,可以用编辑器创建,如最基本的编辑器 Notepad,或用所见即所得的编辑器,如 Adobe FrameMaker,或用结构化的编辑器,如将 XML 文档显示为树状结构的 JUMBO。

生成 XML 文档的数据来源多种多样,可能来自数据库也可能来自 XML 文档,也可能是 Web 上的其他资源。XML-QL、SQLX XQL 等查询语

言及相关技术可用来生成 XML 文档，比如 SQLX（SQLembedded in XML，嵌入 XML 中的结构化查询语言）是一个嵌入 XML 中的 SQL 语言，它为基于 DOM 的应用程序提供了一个存取关系数据库的简便方法。SQLX 接受一个用 XML 描述的查询并把他翻译成一个 JDBC 调用序列，查询结果被转换为 DOM 结构并返回。

3. 解释 XML 文档

XML 的语法分析程序读取文档并检查其中包含的 XML 是否结构完整。如果文档通过了测试，则处理程序就将文档转换为元素的树状结构。目前已有各种语言的多种解析器提供，如 IBM 公司的 XML4 J 和 Sun 公司的 Project X 等。解析有 SAX[4]（Simple API for XML 应用编程接口）和 DOM 两种标准：

（1）在需要详细了解文档的结构、移动文档的组成部分及不止一次地使用文档中信息等情况下使用 DOM；

（2）需要从一个 XML 文档中抽取一些元素、没有很多可用的内存或者文档中信息等情况下，应该使用 SAX 标准。

DOM 提供了更丰富的编程接口。

4. 显示 XML 文档

显示 XML 文档有多种途径。Web 上的 XML 文档资源可以直接显示在 XML 使能的浏览器中；或者使用 XSL 样式表将 XML 文档转换为浏览器能处理的对象，如 HTML 文档；也可以使用程序对 XML 文档进行操作，再将结果按任何一种用户要求的方式显示。再者 XML 技术也支持为同一个 XML 文档生成在多种输出设备上的输出。

7.3 XML 的应用领域

XML 的简单性、开放性、可扩展性、灵活性、自描述性等特性，使其在数据和信息管理数据交换、Web 应用、电子商务应用集成等诸多领域有着重要

用途。

7.3.1　XML 在异构数据集成中的应用

企业在发展过程中必然会积累大量异构数据源,同时,网络的发展又使不同企业不同结构的信息交互成为必然,如何以统一的方式实现这些分布异构数据间的相互识别、相互转换集成,使企业从巨大的数据资源中获取所需的信息,已成为目前企业信息化所要解决的主要问题之一。因此,迫切需要在异构环境下建立统一的集成平台来支撑分布异构数据的集成交互。该集成平台可以屏蔽各种体系结构的异构性,提供访问异构数据的服务,不需要改变底层数据的存储和管理方式,即可实现分布异构数据的互操作;同时,企业业务逻辑可以封装在集成平台的组件中,当业务逻辑发生变化时,只需对该组件进行修改即可,增强了系统的灵活性和适用性。

只有将这些异构的数据源集成起来,提供给用户一个统一的视图,才能从这些资源数据中获取人们所需要的信息。而为大量各种各样的数据提供某种统一的表示方法无疑是解决问题的关键,这就要求能找到一种标准、开放的数据结构来表示数据信息。XML 的出现无疑为异构数据源的集成带来了新的希望。

在网络环境应用中,以企业的数据交换区为核心,建立信息交换集成平台,作为数据传输通道[5-6]。其中所有企业应用系统间需交流的信息以 XML 数据文件的形式存储在数据交换区,进而实现不同应用系统间信息的发布、获取、协商及交换等功能。如图 7-3 所示。并且,需要建立专用的数据库统一管理这些 XML 数据文件。各企业或各企业应用系统间,就是通过该信息交换集成平台,将可通过转换接口生成标准结构 XML 数据文件的各异构数据源联系起来,从而实现数据的集成和共享。

在该方案中,系统逻辑被分为三层:其中数据库层为通道内部已经采用的各个企业提供的数据库;接口层主要是把通道内部数据库的数据经过处理转化为已定义格式的 XML 文档,其处理流程如图 7-4 所示;XML 层则为通道内部之间、通道与互联网之间交换的信息。

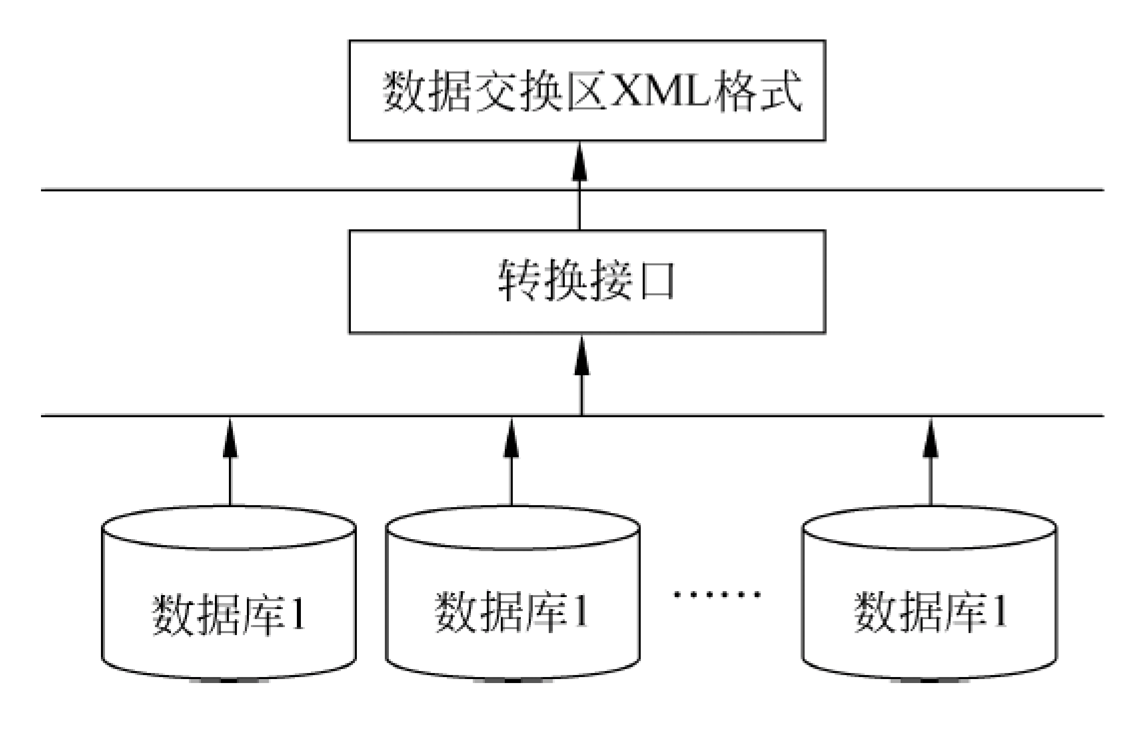


图 7-3 应用集成框架

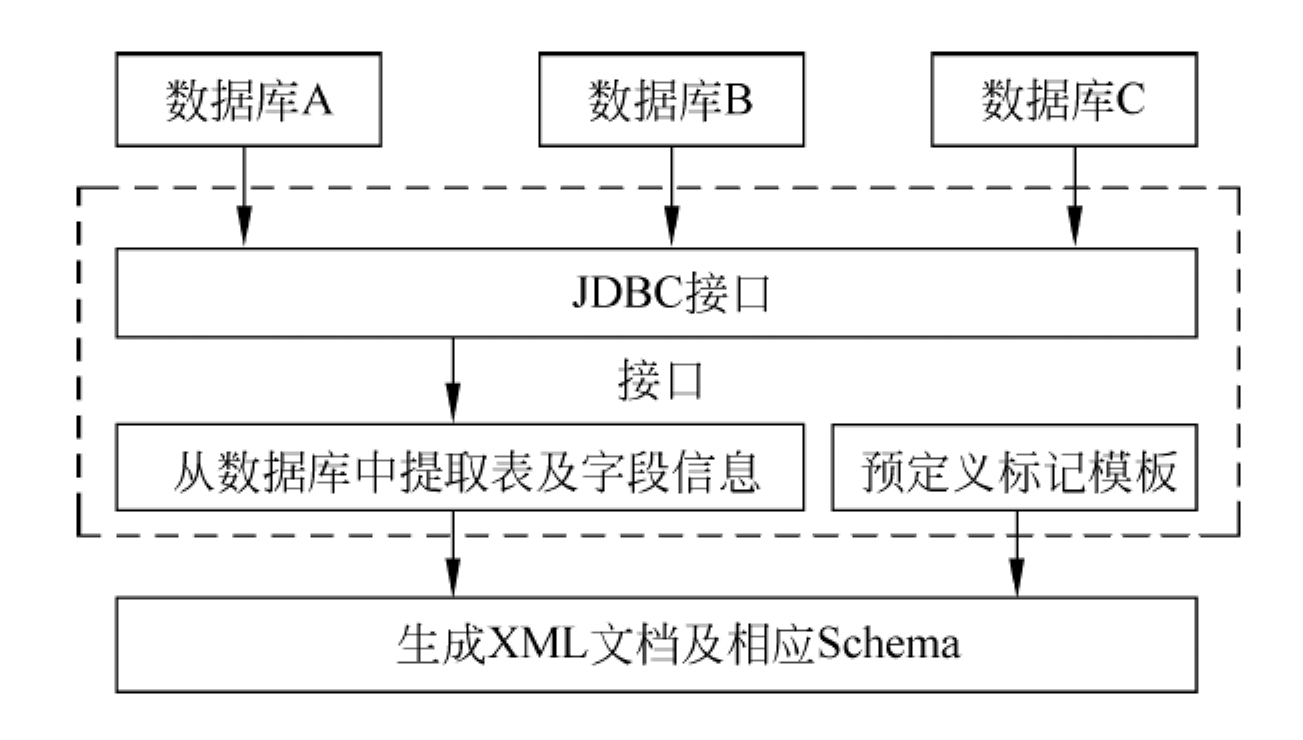


图 7-4 接口层的处理流程

在转换接口中,利用 JDBC 接口建立与数据库的连接,然后从 JDBC 接口中得到数据库中表的基本信息及其表中各数据字段信息,再与预定义标记模板结合生成所需的 XML 文档及相应的 Schema,从而完成从数据库中导出数据为 XML 文档。

7.3.2 XML 在电子商务中的应用

根据电子商务参与方式的不同,电子商务模式主要分为企业对用户(B2C)和企业对企业(B2B)。其中 B2C 类电子商务主要应用于商品的零售业。包括面向普通消费者的网上商品销售(网上购物)和网上电子银行业务(存款业务、取款业务、货币兑换业务等);B2B 类电子商务主要用在企业和企业之间进行的产品采购(包括企业和政府之间的商品采购)和企业与银行之间的银企对账等。

现实中的电子商务发展并不完善，当 B2C 发展到一定程度，发现 B2B 发展的滞后性阻碍了电子商务的进一步发展。而 B2B 的发展又受到现有电子商务技术和标准的束缚。比如 EDI 应用的局限性、行业缺乏数据交换标准等。因此发展 B2B 将是一个战略性的转变，企业网络将向外延伸，与外界建立广泛的交流合作。企业不仅可利用 Intranet/Internet 向广大用户公布信息为客户提供信息查询、货物追踪服务，还能与供应商、销售商、服务商等业务伙伴随时随地保持联系并展开合作。这一阶段运用 XML 技术，可使 B2B、XML/EDI、供应链、客户管理等技术将得到迅速的发展。

数据交换技术对于电子商务起到了推波助澜的作用，到目前为止，它的发展大致经历了三个阶段：

第一阶段，以 EDI 技术为基础的电子数据交换阶段；

第二阶段，以多层结构 WEB 技术为基础的信息发布阶段；

第三阶段，以 XML 等第二代 WEB 技术为代表的数据交换阶段。

作为电子商务的数据交换技术，XML 典型的电子商务应用模式如图 7-5 所示，从图中可以看出，以 XML 为代表的电子商务的数据交换技术涉及 XML 的生成、编辑、解析、转换、浏览和存储[7]。

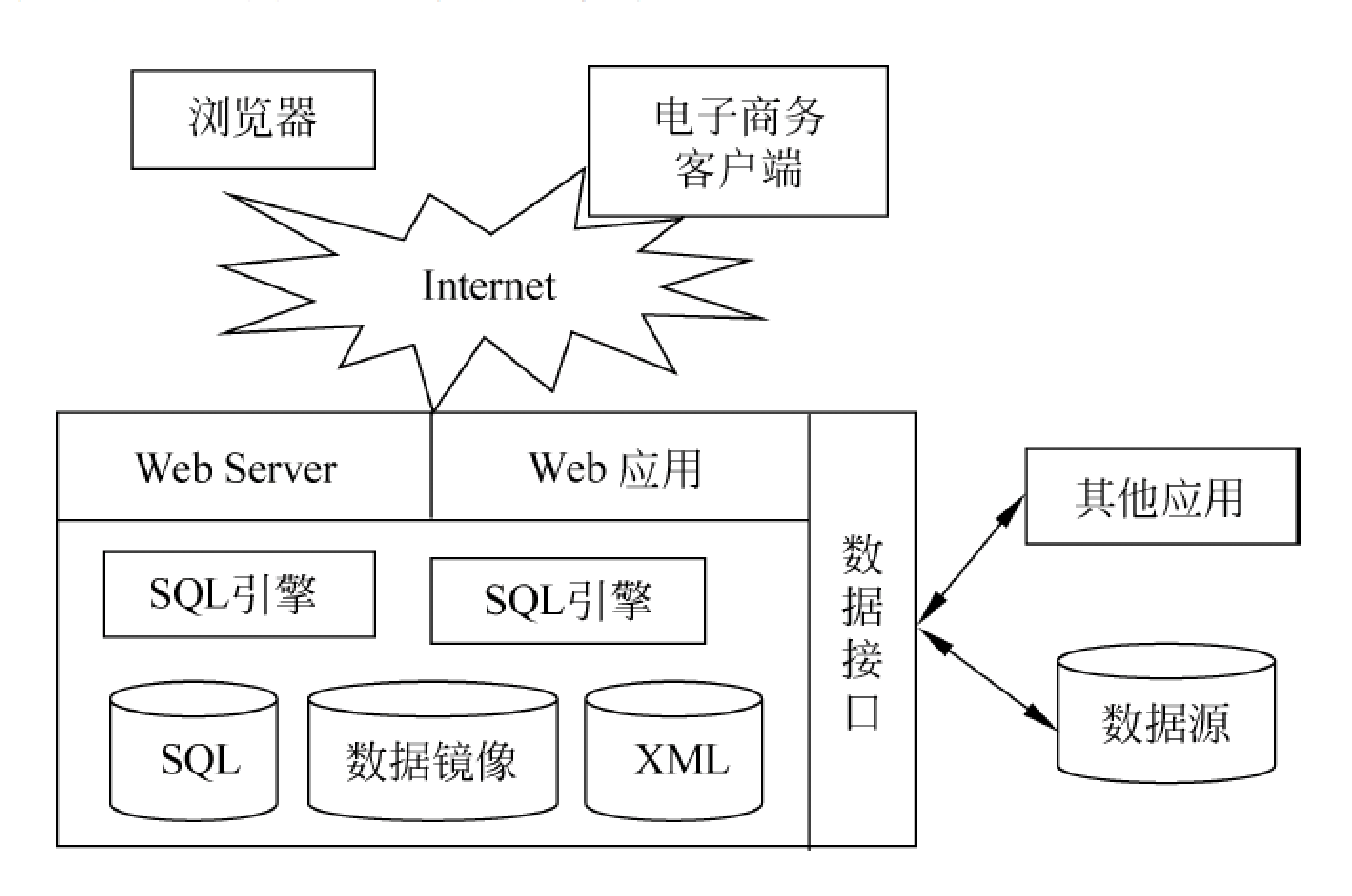


图 7-5　XML 在电子商务的应用模式

XML/EDI 的发展有其必然性，主要是因为传统的 EDI 系统部署和实施起来太复杂，成本也太高。XML/EDI 是通过使用 SMTP 和 FTP 来进行数据格式转换的，在进行数据交换前一般是贸易伙伴拨号到各自的 BBS 上来

启动交换。这种做法会导致供应链关系的混乱，并且连接的成本也过于昂贵。另外，采用 XML 技术的 EDI 有如下优势：

1. 厂商的支持

Microsoft 正在建议 XML Data 方案，快速开发 XML 应用；Microsoft 的 IE 浏览器以及 Netscape Web 浏览器支持 XML；Sun 公司考虑将此标准作为用于 Java 的可移植数据语言。

2. EDI 企业的机会均等

传统 EDI 对较大的公司有利，他们规定商务模式和标准，可以享受 EDI 带来的全部好处，而中小企业被动地接受 EDI，与实现 EDI 方（较大公司）的机会是不均等的。XML/EDI 则不同，它能让所有的参与者都从 EDI 中得到好处，因此它是对称的 EDI。XML/EDI 将为中小企业提供从事电子商务的机会，从而扩大 B2B 电子商务的规模。

3. 传统问题的解决

传统 EDI 过于局限于古板严格的标准。而采用 XML 实现 EDI，实现商业规则与数据分离开来，各个企业可以灵活的开发适合自己的商务应用，能集中于数据内容和结构的交换运用各自的商业规则。许多问题就迎刃而解。

4. Web 风格的 EDI

XML 所采用的标准技术已被证明最适合 Web 开发，应用于 Internet EDI，则可以得到真正 Web 风格的 EDI —XML/EDI。XML 支持结构化的数据，可以更详细地定义某个数据对象的数据结构，这种 XML 数据很容易按关键字进行排序，使查询更为方便。

虽然 XML 未必能解决传统 EDI 的所有问题，但专家们正积极地将它应用到 EDI 的数据管理和数据库管理中。如今 XML 技术已经开始抢占 B2B

市场，可望取代传统的 EDI 技术。

B2B 的电子商务过程是一个将买方、卖方以及服务于他们的中间商（如金融机构）之间的信息交换和交易行为集成到一起的电子运作方式。基于 INTERNET 的 B2B 电子商务显然和在私有网络或者增值网上运行的传统 EDI 有相当的区别。其中最主要的原因是将一个小范围的、局部的、专一的、昂贵的商务概念推广到一个开放的、公众化的、普适的、廉价的系统当中。为了适应这种转变，在 INTERNET 上实现 B2B 电子商务必须具备一定的基础，其内在需求主要表现在信息的标准化和网络交易数据集成技术。而 XML 的 B2B 电子商务很好地解决了这一难题。

（1）XML 若干良好的特性，使它成为 B2B 电子商务的国际性语言。以 Internet 通讯协定（如 HTTP）为基础的 XML，不但其部署的方式较为简易，而且成本更为低廉，所以是没有信息部门的中小企业在部署 EDI 之时最佳的选择。可以肯定，XML 将成为企业数据集成的最佳方法。

（2）XML 将改变企业与企业之间通信方式，并将成为企业数据集成的实际标准。随着企业与企业之间电子商务的发展，需要有进行采购单、零部件、商品信息等方面的信息描述和交换。早期的几种技术，比如 CORBA、EJB、COM 等，都没有能够提供在多个公司的计算机系统之间共享数据的简易方法。

（3）XML 将能取代传统的 EDI 系统，使用 XML 小公司也可以通过来回发送采购单，进行电子商务活动。利用 XML 技术将改变传统的价值链，从而实现供应链的管理。

（4）实现 B2B 电子商务尤为重要的是异构系统间的连接，在进行企业间的联合、兼并时，最重要的课题是企业间系统的集成。最近涌现出了一种称为“EAI（Enterprise Application Integration：企业应用集成）”的软件，作为能平滑、廉价实现系统间连接的技术，它将和 XML 共同支撑 B2B 电子商务强劲、迅猛扩展的势头。

7.3.3　XML 在电子政务中的应用

数据是电子政务系统建设的核心，应成为独立于各个系统、不断重复使

用、能长期保留并在各类政府业务系统之间实现共享、交换和传递的信息资源。信息显示技术就是解决这些信息的规范化问题,可以为任何系统、任何平台提供通用的数据内容显示、定位和查询方法。利用 XML 技术可以解决电子政务数据显示、文档定位、文档查询等基本问题,为电子政务应用系统提供信息表示、消息服务、事务处理等功能,便于电子政务应用系统对信息进行解析、提取、传递、交换。

基于 XML 技术的电子政务应用标准体系[8]见表 7-1。

表 7-1 基于 XML 技术的电子政务应用标准体系

电子政务应用	功能作用	适用 XML 技术标准
应用支撑层	数据访问	DTD、SCHEMMA、DOM、SAXXSLT、XPATH 、XQUERY、XPointer、XLINK、XQL
	事务处理	BTP、WS-Coordination、WS-Transaction
	流程控制	WPDL、XPDL、BPML、WSFL、BPELAWS、XLANG、ebXMLBPSS、WSCI
	信息交换	DTD、XSLT、SCHEMMA、WSDL、XML-RPC、DOM、SAX
	目录服务	DSML
应用层	公文处理	DTD、WSFL、SCHEMMA、WPDL、BPML、XSLT、XPATH、Xpointer、XQUERY、XLINK、XQL、DOM、SAX、BTP、WS-Coordination、WS-Transaction、XLANG、XPDL
	业务处理	ebXMLBPSS、BPEL4WS、BTP、WS-Coo-rdination、WS-Transaction
	信息发布	XHTML、RDF、RSS、CSS、XSLT、WML、P3P
	信息采集	DTD、SCHEMA、Xforms、XSLT、SOAP、XMLSECURITY
	服务调用	UDDI、SOAP、WSDL、WSCL
信息安全	XMLSignature、XMLEncryption、XKMS、XACML、SAML	
系统管理	DTD、SCHEMA、XML-RPC、XLANK、XPATH、Xpointer、XSL、XQUERY、DOM、SAX、XQL、BTP、WS-Coordination、WS-Transaction	

电子政务系统除了涉及数据资源的描述及显示的 XML 技术外,还拥有与其他应用不同的安全保护方法技术。政务系统涉及电子公文等大量的政务数据,有不同级别的机密性和完整性要求,要保证数据在存储、处理和传输时不会泄露和不被篡改;有互联互通和业务协同的要求,要保证电子政务应用系统能够确认访问用户的合法性。安全服务是在信息安全基础设

施上跨越电子政务各应用系统和层次，为电子政务应用提供安全可信服务，主要涉及鉴别与身份认证、授权与访问控制、数据加密、数字签名、审计与日志等内容。基于 XML 的电子政务安全技术包括 XML Signature、XML Encryption、XKMS（XML Key Management Specification）、XACML（eXtensible Access Control Markup Language）、SAML（Security Assertion Markup Language）等技术规范，主要有授权（确保身份不被假冒）、完整性（确保信息不被篡改）、隐蔽性（确保文档不被窃取）、不可否认性（确保文档的不可否认）、阻塞预防（确保高复杂度的操作）等功能和作用。

1. XML Signature 签名方法

XML 签名用来确保文档的完整性、授权性和不可否认性。XML Signature 允许特定的人按特定的顺序对文档的特定部分进行签名，允许对已经加密过的其他部分进行进一步加密。需要注意的是：由于数字签名使用了特定专用密钥来认证，当签名人以纯文本形式查看文档时，可能对其他已经加密过的信息进行了解密。XML 的规范性特点要求：如果对应用了密码散列算法的信息进行最轻微的更改，将会产生不同的值。这一点为信息的完整性提供了信任，但也可能会产生逻辑结构相同、确切文本不同的两个 XML 文档。XML 规范描述了一种生成文档的范式，用来解释具有同一范式的两个文档。因此，在生成签名和验证计算时，要在范式上进行信息摘要，如果摘要匹配，即使文本形式不同，也认为范式匹配。签名与签名对象或文档是分离的，本身并不参与计算或验证。

2. XML Encryption 加密方法

XML Encryption 用来确保文档中敏感信息的隐蔽性。XML Encryption 允许用不同的密钥，根据业务的需求对文档中敏感的信息进行加密，允许将相同的文档发送给不同的接收者，而接收者也只能将与自己相关的信息进行解密。XML 加密语法的核心元素是 Encrypted Data 元素，包含 EncryptionMethod、Keyinfo、Cipher Data 三个子元素；Encryption Method

定义了加密的算法，如 AES、3DES、RSA、RCA 等；Keylnfo 提供或指明了加密密钥；Cipher Data 提供了密文。对任意数据加密时，Encrypted Data 元素成为新 XML 文档的根或成为一个子代元素；对整个 XML 文档加密时，Encrypted Data 元素成为新 XML 文档的根；对元素或元素内容加密时，Encrypted Data 元素替换 XML 文档加密版本中的该元素或内容。

3. XKMS 注册方法

XKMS 协议定义了 XML 签名协议中所需公共密钥的分发和注册方法，由 X-KRSS(XML Key Registration Service Specification)和 X-KISS(XML Key Information Service Specification)两部分组成。X-KRSS 是对密钥对的拥有者进行注册的协议，已经注册的密钥对用来通过 X-KISS 协议；X-KISS 是对 XML 签名、XML 加密以及其他密钥信息进行处理的协议。

4. SAML 授权方法

SAML 使用 OASIS 维护的认证、授权决定和属性三种"断言模式"语句，来决定谁是请求者、请求什么、是否接收请求，并处理实际授权和认证请求和反馈信息的实际交换。一个 SAML 请求包含了认证用户名和密码、有关请求发起方的详细信息，这些信息按所设计的进程进行处理，并用 XAML 来决定是否接收对 XML 资源的访问。SAML 最主要的特点是采用点到点的信任模式，实现了单点登录多个系统的可能性。

5. XACML 控制方法

XACML 与 SAML 提供了一种标准的存取控制决策方法，XACML 接收到一个 SAML 请求时，根据规则集合、策略来决定是否允许对某个特定 XML 文档的访问。XML 文档中的 tag 所包含的 XPointer 和 Xpath 表达式将通知分析器评估 XACML 策略，评估后由 SAML 授权作断言处理。

7.3.4 XML 在网络管理中的应用

随着互联网的飞速发展，网络规模不断扩大，网络复杂性不断增加，对

网络管理的要求也越来越高。传统的网络管理是基于 IETF 在 1988 年提出的 SNMP 协议及其相关规范。SNMP 以其简单和实用性已经成为事实上的工业标准,形成了当前普遍基于 SNMP 协议的网络管理框架。但是,由于 SNMP 协议本身固有的问题,这一框架在信息模型、协议安全性、传输协议等方面有许多无法克服的缺陷。因此,SNMP 已不能对日益复杂的现代化网络进行有效的管理,在配置管理方面的缺陷尤为明显,人们迫切需要新的网络管理模型。

可扩展标记语言 XML 的出现为构建新的网络管理模型提供了契机。目前学术界对于 XML 技术在网络管理中应用的研究大致可以分为三类。

第一类是 XML 和 SNMP 集成管理,如 J. P. Martin-Flatin 提出的 SNMP MIB 到 XML DTD 文档的转换模型[9],F. Strauss 开发的"libsmi"工具[10];Avaya Labs 用来读写 SNMP agent 中的管理信息的 XML 接口[11]。

第二类是基于 XML 的管理架构,如 J. P. Martin-Flatin 在文献[9]中提出的 WIMA 和 Hong 等提出的基于 XML 和 HTTP 协议的 XML-based Management 架构[12]。

第三类是采用 Web Services 技术来进行基于 XML 的网络管理,这方面的代表是 IRTF 的 NMRG 和 OASIS 这两个组织。

文献[13]提出了一个 XML 到其他协议的转换网关方案,该方案既能管理传统的基于 SNMP 代理的网络设备,又能管理基于 CLI(Telnet/SSH)的设备,基于这一网关实现了一个既能监控网络又能配置网络的配置管理系统。

XML 协议转换网关的主要任务是:把 HTTP 协议传输的 XML 请求转换为相应的 SNMP 操作或者 CLI 命令,并把得到的响应也编码成 XML 文档,通过 HTTP 协议返回给请求者。一方面,网关接受包含 XPath 表达式的 HTTP GET 请求,对请求进行解释,生成 SNMP 请求发送给设备。在收到响应之后,应用 XPath 表达式进行过滤,得到请求者需要的数据;另一方面,请求者以 HTTP POST 方式向网关发送配置请求的 XML 文档,网关将请求翻译 CLI 命令,通过 Telnet/SSH 协议登录到远程设备上执行,并把响

应编码成 XML 文档返回给请求者。网关的结构及数据流程如图 7-6 所示，具体实现参阅文献[13]。

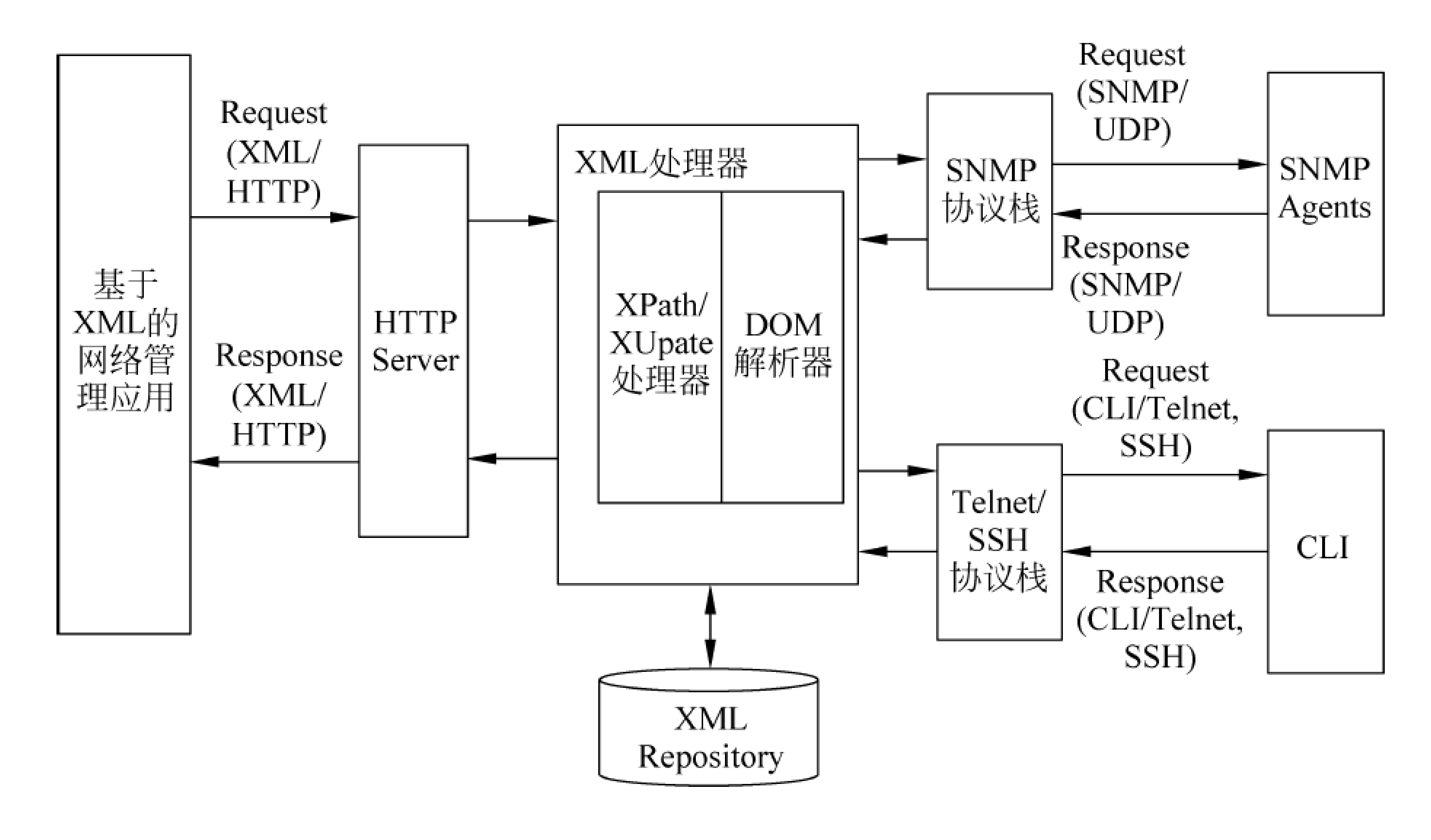


图 7-6 系统架构及数据流程

7.4 信息资源的表述和发布技术

2007 年 5 月 W3C 关联开放数据项目正式起动，其目标是号召人们将现有的数据公布成关联数据，并将不同数据互联起来。关联数据提出的目的是构建一个计算机能理解的具有结构化和富含语义的数据网络，而不仅仅是人能读懂的文档网络，以便于在此基础之上构建更智能的应用。目前，关联数据逐渐得到学术界、工业界、政府部门的广泛关注。自 2006 年以来，关联数据应用已扩展到网络通用本体、大型传媒、商业企业、政府部门、图书馆、医药学、农业信息等众多领域。目前比较知名的大型应用有美国国会图书馆及其主题标目（LCSH）、瑞典国家图书馆 LIBRIS 国家书目、英国广播公司（BBC）的关联数据应用、纽约时报等。国内的关联数据研究尚处于初级阶段，大型应用较少，但也出现了一些较好的实际应用项目，比如宁波市数字图书馆服务外包产业信息门户（SOIP）。

7.4.1　关联数据的基本原则和特征

关联数据的概念为 WWW 的发明者，被誉为互联网之父的 Tim Berners-Lee 于 2006 年在《关联数据笔记》中首次提出，在该文中他分析了 Web 的发展与演变，提出了发展数据网络的思想，而数据网络的核心和关键则是关联数据。2009 年在 TED 大会上，他提出关联数据就是一箱箱数据，当通过开放标准关联在一起时，从中可以萌发出很多新事物和新应用。有的学者侧重对语义的认识，如白海燕认为关联数据是用来在语义网中使用 URI 和 RDF 发布、分享、连接各类资源，强调建立已有信息的语义标注和实现数据之间的关联，具有框架简洁、标准化、自助化、去中心化、低成本的特点，为构建人机理解的数据网络，提供了根本性的保障，为实现语义网远景奠定了坚实的基础。Boutin G 也持同样的观点，认为关联数据是提供了关联结构化数据的新媒介，可以更好地让机器读取这些数据。还有一些学者则认为关联数据是一类实践活动。维基百科的定义：关联数据是一种推荐的最佳实践，用来在语义网中使用 URI 和 RDF 发布、分享、连接各类数据、信息和知识。Christian Bizer 也认为关联数据是利用网络关联不同类型数据的实践[14]。

1. 关联数据的基本原则

Berners-Lee 提出的关联数据遵循四个方面的基本原则，获得了业界的广泛认同：①使用 URI 作为任何事物的标识名称；②使用 HTTP URI 让任何人都可以访问这些标识名称；③当有人访问某个标识名称时，提供有用的信息(采用 RDF、SPARQL 标准)；④尽可能提供相关的 URI 链接，以使人们可以发现更多的信息。

IBM CSDL Web2.0 开发人员张静、马春娥经过分析也提出了构建和实现关联数据的三原则：①资源。发布一个领域的数据之前，要确定发布的资源是什么。只要你认为是有意义的，有被引用必要的，都可以称为资源。②资源标识。任何一个资源都是用 HTTP URI 来标识，之所以要用 HTTP

URI 来标识，是希望数据能够通过 HTTP 协议访问，真正实现基于 Web 的访问与互联。③资源描述。资源可以有多种描述，如 HTML、XML、RDF 以及 JPEG。文档 Web 的文档主要通过 HTML 格式来表示，数据 Web 的数据主要通过 RDF 格式来表示。RDF 将一个资源描述成一组三元组（主语、谓语、宾语）[15]。

2. 关联数据的特征

关联数据是在网络上使用资源描述框架（RDF）作为数据的模型，运用统一的资源标识符（URI）作为数据的标识，通过 HTTP 协议协调规范“数据”，从而实现在网络上的发布。这样发布的数据揭示了数据间的相互联系，提供了计算机可以理解的词义信息[16]。

关联数据可以通过数据间的链接形成关联的数据网络。关联数据的数据之间的链接多种多样，既能在来源不同的数据库之间建立链接，又能在不能相互操作的数据之间建立链接。而且，关联数据既能连接其他数据，也能被其他数据所连接。因此，具有连接的多样性和广泛性。

关联数据网络有异于超文本网络，其相同之处在于都是链接，不同之处在于超文本网络是把 HTML 文件通过超链接连接在一起，关联数据不是单纯地链接这些文件，而是采用 RDF，建立关联数据网络，通过数据描述，链接世界上所有事物的数据网络。

7.4.2　关联数据的发布

发布关联数据的途径往往因资源内容的特征被分成三种[17]：如果数据量很小（几百条 RDF 三元组或者更少），可以直接采用静态的 RDF 文件（静态发布）；如果数据量很大，则需要将它们放进 RDF 库中，并选择 Pubby 服务器作为关联数据服务的前端；如果数据的更新频率很大，就需要引入更新机制，或者在请求数据的时候再根据原始数据在线生成（on-the-fly translation）RDF。其中的第三种方式，即在线映射，往往会借助于一些映射工具[18]，如：D2RQ 平台、Virtuoso RDF Views、Sparqlify 等。此外，W3C 还

有一个 RDB2RDF 工作组，从事 R2RML 映射语言的研究。

很多传统的信息都借助于关系型数据库进行存储，另外由于 D2RQ 的使用简单方便，因此 D2RQ 在很多场合都得到了应用。作为关联数据发布的标准教程，文献“How to Publish Linked Data on the Web”就重点推荐了 D2RQ，并介绍了它的软件架构和使用方法。D2RQ 平台包括 D2R Engine、D2RServer 以及 D2RQ 映射语言，基于 D2R Server，目前已经有很多数据源对外开放了关联数据的接口（即变成了关联数据的数据集），如 DBLP 书目库、CIA Factbook、欧洲国家地区统计信息库等。

采用 D2R Server 将关系型数据发布成关联数据，软件操作流程如下：

(1) 准备 Java 环境，下载某个版本的 D2R Server，如：d2rq-0.8；

(2) 执行 generate-mapping 工具，连接至数据库，生成映射文件，如：mapping. n3；

(3) 根据发布需求，在以上生成的 mapping. n3 文件基础上进行修改与完善；

(4) 以 mapping. n3 为参数，启动 d2r-server。

经过以上简单四步，D2R Server 即可提供关联数据的访问服务，这些服务包括：用户可以浏览某类实体的列表(directory)，并通过每个 URI 访问到某一条实体的网页描述和 RDF 描述。同时，D2RServer 还提供了 SPARQL 查询接口，并提供了一个 Web 化的执行界面(SPARQL Explorer)，用户可以在该界面输入 SPARQL 查询语句，并查看到执行结果。

作为知识的不同载体，科技文献与科学数据无论在内容上还是在语义描述模型上都有各自的特点，另外科学数据还带有强烈的学科领域特征。文献[18]以科技文献和科学数据的发布为例，研究了基于 D2R Server 发布关联数据的流程和其中的关键问题，值得参阅。

仅从技术上看，关联数据的实现并不复杂。但是，要让当前互联网上海量的基于超链接的“文档的网络”转换成基于关联数据的“数据的网络”，是一项浩大的工程，需要简便可行的模式和方法。众多的内容管理系统(简称 CMS，下同)是目前网络资源发布的主力，其中 Drupal 是一款技术领先、用

户众多、特色鲜明的开源软件，它基于完全开源的 XAMPP 架构，具有高度模块化、优良的用户友好性等特点，并拥有庞大的网络社区支持，成为 Web2.0 时代的翘楚和 CMS 中的佼佼者。难能可贵的是，它很早就开始了对语义网技术的支持，使得它在新时代得以引领潮流。2009 年，Drupal 已形成了一整套支持关联数据发布(publish)和消费(consume)的完整机制[19]。

7.5 基于关联数据的知识发现模型

知识发现是人类的主要知识活动之一，当前的知识发现活动也越来越多地基于网络数据资源环境。在网络资源环境向“语义网”方向前进，并已经进入“关联的”数据网络时代的时候，知识发现必然面临着新的机会和挑战；与此同时，知识发现也是关联数据网络发展和完善的主要动力。因此，基于关联数据的知识发现是以关联数据网络资源环境为基础，发挥关联数据带来的优势和潜力，发现可理解、可用的新知识所必要和必需解决的问题。

从知识发现研究角度来说，基于关联数据的知识发现是知识发现的特殊案例。广义的知识发现关注于从数据中发现知识的整个过程，包括数据是如何存储和访问，算法如何自动处理数据并且在大量数据的环境下有效运行，结果如何进行解释和可视化，以及整个过程中人机交互如何建模和支持。关联数据本身是应用新方法收集、组织和存储的数据资源。虽然关联数据与以往的关系型数据库在数据结构、关系类型、结构化程度、语义等方面存在很多不同，但区别于文件网络(Web of Document)，关联数据的目标和特征就是提供一个全球共享的超级数据库(见图 7-7)。因此基于关联数据的知识发现活动应当是在遵守数据库知识发现的一般规律的同时，考虑数据组织方式、应用技术、资源环境的变化特殊性的特殊案例。

从关联数据的应用角度来说，知识发现是关联数据的一种关键和高层的应用。关联数据的广泛应用和数据网络的飞速发展，已经宣布了面向语义网的信息收集、组织、存储和访问的新时代的到来。以关联数据为基础的

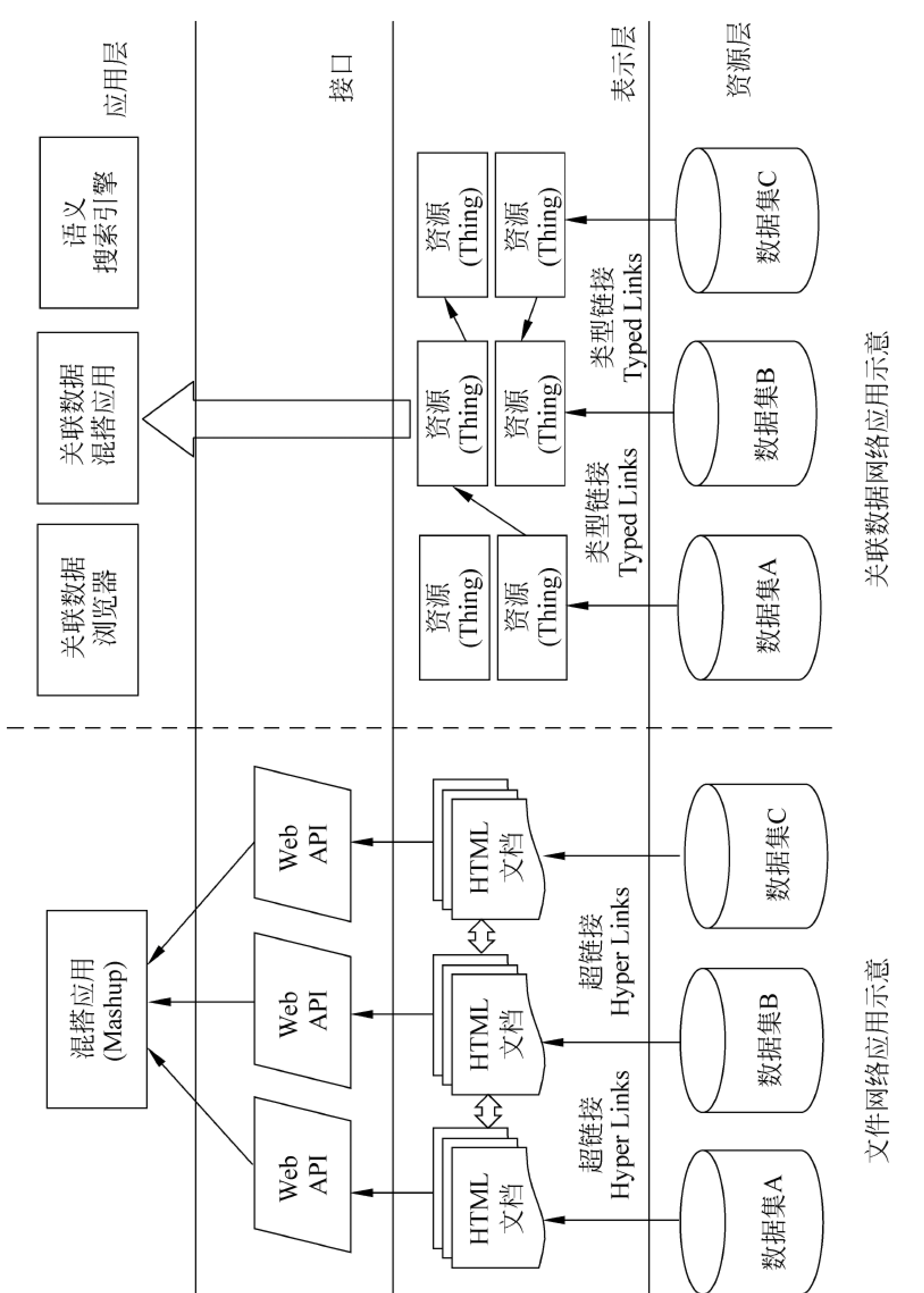


图 7-7 文件网络和关联数据网络应用的对比

数据对象、网络环境、语义关系模型、存取标准(HTTP URI)和网络应用(浏览、搜索等)为更多的网络应用提供了新的可能。如何根据关联数据的特点,发挥关联数据的优势,帮助人们更容易、更准确、更全面、更高效地发现所需要的信息,最终获取新颖、有用并且可用的知识是关联数据应用研究和开发的主要方向,因此,基于关联数据的知识发现是关联数据的关键应用和研究方向之一。

7.5.1 基于关联数据的知识发现的潜力和特征

关联数据提供了一个更通用的,更灵活的出版范式,它使得数据消费者更容易发现和整合来自大量的数据源的数据,关联数据提供以下的机制(见图 7-8)。

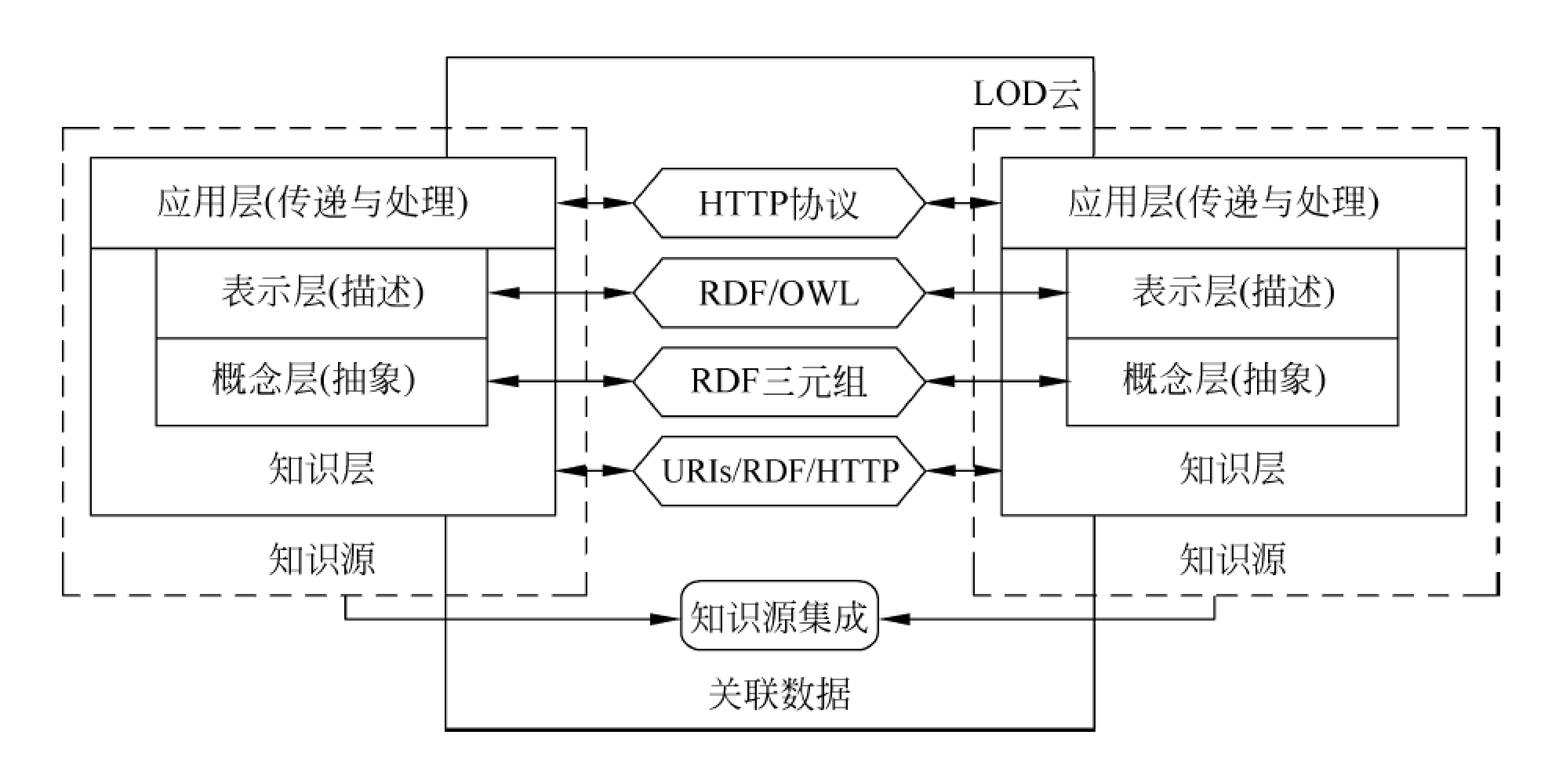


图 7-8 关联数据的知识源集成能力

统一的数据模型:关联数据依赖 RDF 作为一个单独的、统一的数据模型。通过提供实体的全球统一标识符并且通过允许不同的规范格式并行地用于数据表示,RDF 数据模型已经特殊地为全球数据共享的案例设计。相反,其他的网络数据发布方法依赖于大量的不同数据模型,并且导致了在整合处理中异质性桥接的需求。

标准的数据访问机制:关联数据承诺自身使用 HTTP 协议的特殊模式。这一协议允许数据源使用通用的数据浏览器访问,并且允许完整的数

据空间使用搜索引擎被爬行。相反，Web APIs 使用不同所有者的界面访问。

基于超链接的数据发现：通过使用 URIs 作为全球实体标识符，关联数据允许超链接被设置在不同数据源的实体之间。这些数据连接所有关联数据到一个单独的全球数据空间，并且允许关联数据应用程序在运行时去发现新的数据源。相反，Web APIs 和使用所有权形式的数据 dumps 仍然是孤立的数据岛屿。

自描述数据：关联数据依靠共享的词表简化了不同数据源的整合，通过参引术语 URIs 使得这些词汇的定义可检索，并且通过词表链接允许来自不同词汇表的术语被相互连接。

全球开放共享：关联数据为实现高层次科学数据应用提供了新的机会。越来越多的领域采用关联数据最佳实践，逐渐地形成了统一框架、标准访问、语义关联和开放共享的环境，建立了一个全球数据共享基础设施，成为所有学科共享科学数据的范式。

1. 基于关联数据的知识发现的潜力

（1）具备以上机制的关联数据，其主要优势在于以一种标准、简捷、去中心化和低成本的方式在网络的数据层提供标准规范和简便的整合机制，为其基础上的知识发现提供了新的潜力。

（2）扩展了知识发现的范围：关联数据的基本数据特征的理论优势，将使得知识发现得以在不必关心资源的物理位置、访问接口和数据形式的平台上进行，为知识发现提供了更广阔的领域和更多的机会。通过关联数据在多个数据集的数据之间的链接，人们得以发现更多的意料之外的资源。

（3）增强了知识发现的能力：在文档网络中，无论是 APIs 或 Web APIs，都需要根据应用需求进行定制开发。当应用发生变化时，必须调整或者重新开发接口程序。在网络应用程序和信息环境不断发展和变化的情况下，这一任务越来越不可持续，极大地影响了高层应用的有效性和稳定性，数据仓筒、信息孤岛的问题并没有得到真正的解决。而关联数据则因为其

自身的特征，其顶层应用可以不需要中间接口层，直接浏览、搜索、获取和混搭应用数据资源，保证了应用的动态性、完整性和可用性，增强了知识发现网络应用的能力。

由此可知，基于关联数据的知识发现，有可能真正地打破知识在物理上和逻辑上的分割和独立，更广泛地发现知识，更精确地选择和组织知识，更规范地揭示知识，更好地在广泛、动态和完整的基础上完成知识的发现和创新。

2. 基于关联数据的知识发现的特征

基于关联数据的知识发现在关联数据的数据、数据源、网络资源环境和应用规则的特征基础上，从应用角度看，基于关联数据的知识发现特征主要体现在：

（1）知识发现在网络规模上实现

知识发现的范围被扩展到由异质、异构、分布式知识源相互关联形成的全球数据空间上，知识发现活动得以在一个网络规模上实现。

（2）实现语义化的知识发现

关联数据采用机器可读的结构化语义数据模型，因此在数据处理中可以由机器自动和高效率地理解和处理显性描述的数据间的语义关系，使得知识发现可以在语义查询能力的支持下，通过已经存在的语义关系，发现相关的资源，并且允许推断和发现资源之间的进一步关系，乃至形成语义关联的新知识。

（3）知识发现的动态扩展和知识动态求精

关联数据开启了新的应用可能，允许了动态可扩展的知识发现。去中心化和标准访问机制使得人们不必完全了解数据源底层结构和语义查询底层逻辑，并且可以适应数据源的动态性，允许在任务执行的期间及时发现新的知识源，从而提高了知识发现的准确性和完整性。

7.5.2 基于关联数据的知识发现过程分析

基于关联数据的知识发现是关联数据的高层应用，是在关联数据理论

框架、技术和资源环境的基础上的创新性知识活动。

基于关联数据的知识发现过程遵循知识发现的一般活动规律，同时因为技术架构和网络资源环境的变化而有其独特性。其过程由知识发现一般过程比较和推论如下（见图 7-9）。

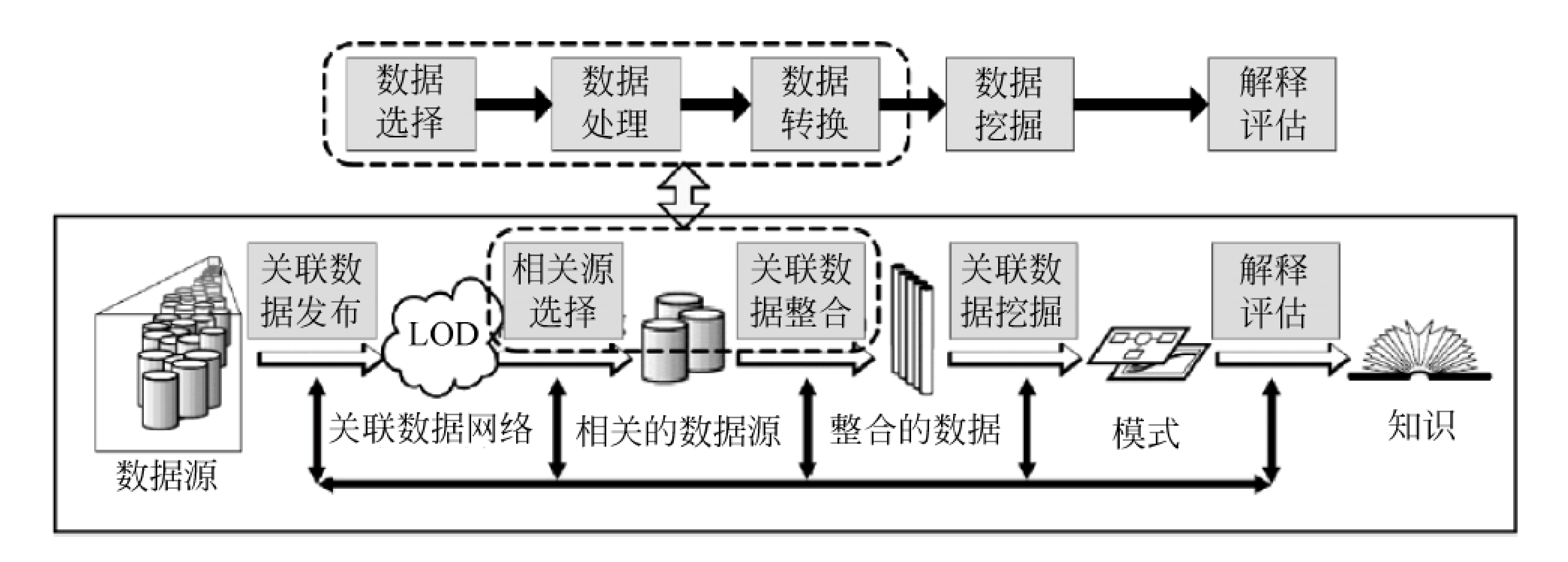


图 7-9　基于关联数据的知识发现过程

基于关联数据的知识发现过程包括了关联数据发布、相关源选择、关联数据整合、关联数据挖掘等基本阶段。

1. 关联数据发布

与以往数据库知识发现不同的是，基于关联数据的知识发现是扩展到整个网络规模的，知识发现的基础是关联数据网络。因此对于整个基于关联数据的知识发现综合过程，首要的是知识发现的用户融入到关联数据网络中。这种融入可以包括发布自己的数据集为网络上可用的数据，以及构建数据集内部和数据集之间的关联关系。这一过程的底层逻辑是关联数据的发布原则，我们称之为关联数据发布阶段。这一数据准备阶段本身也包含了关联数据的链接发现和构建过程，同时为基于关联数据的知识发现提供了丰富的语义互联的数据源。

2. 相关源选择

关联数据网络中的数据源数量巨大，并且动态增长，它们来自不同的数据提供者，属于不同的领域，采用不同的本体或者词表术语描述概念，使用

不同的访问方式。在这样的海量、异构和动态的数据源上进行知识发现，数据选择是非常重要的步骤。如何能够根据用户查询需求识别和筛选出相关的数据源，同时兼顾完整性、准确性和效率性，是基于关联数据的知识发现的关键问题。

3. 关联数据整合

关联数据的整合包括了从相关的数据源中查找和获取数据、根据需求对获取的数据进行清理以去除不相关的数据、消除数据的不一致和歧义、对结果进行相关性排序、将实体根据规范的表示进行合并，等等。因为关联数据、关联数据网络的特征提供了知识源的透明集成机制，关联数据的自然属性使得传统知识发现过程中的数据选择、数据处理和转换过程得以被替代。关联数据整合通过关联数据解析查询、模式映射、身份识别、质量评估、关联数据合并等特殊功能完成数据的查询、集成、过滤、分割和合并等处理，最终形成待挖掘的整合关联数据。

4. 关联数据挖掘

数据挖掘是知识发现的关键步骤，关联数据挖掘同样是基于关联数据的知识发现的核心过程。关联数据上的挖掘可以分为两个层级的任务，第一个层面是在整合的关联数据集上，调度和运行传统的数据挖掘，完成制定模式的知识发现。这一层级的数据挖掘工作需要考虑到将关联数据的检索过程从数据挖掘过程中分离出来，以便减轻用户使用和学习扩展 SPARQL 和了解关联数据的底层逻辑（本体、语义）的负担；第二个层面的挖掘是利用关联数据自身的特点，去挖掘关联数据网络的最大潜力，即通过链接挖掘与发现关联数据网络中隐藏的、丰富的、潜在有用的关系。这一层级的挖掘工作目标是创建针对关联数据特性的挖掘算法、知识模式以便在已有的语义关联基础上，推断和发现任意资源之间的进一步关联，或者通过特定模式重新组织和架构形成新的关联知识。

7.5.3　基于关联数据的知识发现模型

基于关联数据的知识发现是一个综合的系统工程，在遵守知识发现一般规律的同时，具有自己的特征。根据前述的基本分析，基于关联数据的知识发现可以概括为一个分层的应用模型，将基于关联数据应用的特殊性结合到知识发现的一般规律中，构建出模型[20]，如图 7-10 所示。

从模型中可见，整个知识发现的底层支撑是关联数据的方法、技术、数据和网络环境；资源层是具备良好关联的关联数据网络；知识发现处理层包含关联数据查询与获取的专门过程、数据处理整合过程；应用层是知识发现的用户界面，包括知识发现的问题定义、查询输入、结果输出和评估反馈过程。此外，知识发现本身是一个循环求精的过程，根据结果的评估和反馈，可以循环回到知识发现过程的前续阶段进一步获得更为准确和符合需求的结果。

基于关联数据的知识发现模型从以下几个方面体现出其独特之处。

1. 控制逻辑

基于关联数据的知识发现模型以关联数据方法（数据、技术、资源环境、标准、机制）作为底层控制逻辑。因为在该模型中，知识发现被视为基于关联数据的特殊应用，所以模型的底层控制逻辑遵循基于关联数据应用的一般规律，具有关联数据应用的相关特征。数据准备、数据获取、数据处理和数据挖掘处理过程，都需要根据关联数据的特殊需求进行重新设计和解决。

2. 流程和结构控制

模型以知识发现作为基本方法，程序执行过程和数据的流程自关联数据发布过程向上，完全拟合知识发现的一般过程，最终实现发现知识的最终目标。模型结构也因流程自底向上采用分层结构，该结构也符合基于关联数据的应用框架结构。证明了在此模型中很自然地将关联数据的应用与知识发现方法融合在一起，各自发挥优势，解决科学的问题。

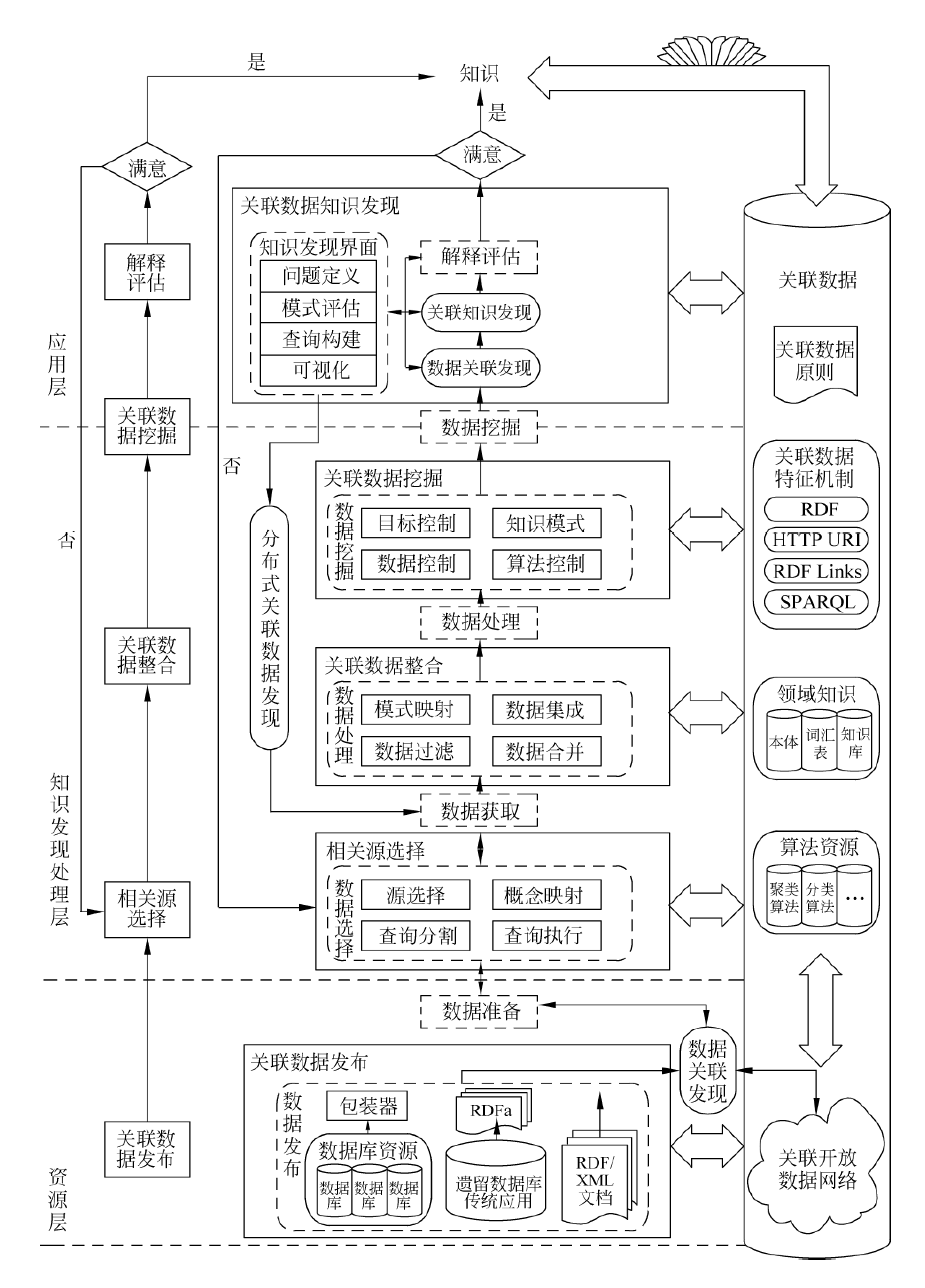


图 7-10　基于关联数据的知识发现模型

3. 关键操作控制

模型的各层之间和各功能模块之间依靠几个关键操作进行衔接、互动和控制，即数据关联构建操作、分布式关联数据发现操作和关联知识发现操作。这些功能和操作实现模型中资源的调用、组织和生成，是基于关联数据的知识发现能否实现的关键。它们主要涉及关联数据的操作，因此问题的解决需要面对关联数据提出的挑战。而这几个关键操作问题的解决，最终将实现知识发现的范围、效率和能力的提升。

参考文献

[1] 瞿裕忠，张剑锋等. XML 语言及相关技术综述[J]. 计算机工程. 2006，26(12)：4-6
[2] W3C. Document Object Model (DOM) Level 1 SpecificationVersion 1. 0. W3C Recommendation，1998-10-01. http://www. w3. org/TR/1998/REC-DOM-Level-1-19981001/
[3] 阎红灿. 面向 Web 的 XML 文档数据管理及分类检索技术研究[D]. 天津大学管理学院. 2009. 2
[4] Megginson D. SAX：The Simple API for XML. 1998-05. http://www. megginson. com/SAX/index. html
[5] 王韦伟，孙庆鸿. 基于 XML 的分布异构数据集成平台[J]. 东南大学学报(自然科学版). 2006，36(5)：715-719
[6] 陈桦，麻风梅，韩艳艳. 基于 XML 的异构数据集成模式的研究[J]. 微电子学与计算机. 2009，26(1)：137-139
[7] 齐华宁，李姝. XML 技术和电子商务的发展[J]. 辽宁大学学报(哲学社会科学版). 2001，29(5)：105-108
[8] 宋彦敏. 基于 XML 技术标准的电子政务应用方法[J]. 图书与情报. 2004. 3：55-57
[9] Flatin J P M. Web-based Management of IP Networks and Systems[D]. Lausanne：Swiss Federal Institute of Technology，2000
[10] Strauss F. A Library to Access SMI MIB Information[EB/OL]. [2006-12-10]. http://www. ibr. cs. tu-bs. de/projects/libsmi/
[11] Avaya Labs Research. XML-based Management Interface for SNMP Enabled Devices[EB/OL]. (2001-09-08). http://www. research. avayalabs. com/user/mazum/Projects/XML/
[12] Ju H，Choi M，Han S，et al. An Embedded Web Server Architecture for XML-

Based Network Management[C]//Proc. of IEEE/IFIP Network Operations and Management Symposium. [S. l.]: IEEE Press, 2002
[13] 章勋,杨家海,王继龙等. 基于 XML 技术的网络配置管理系统[J]. 计算机工程. 2008,34(3): 127-129
[14] 肖强,郑立新. 关联数据研究进展概述[J]. 图书馆学理论研究, 2011.7, 55(13): 72-75
[15] 张静,马春娥. 如何利用 D2R 发布 linked data. [2010-09-26]. http://www.ibm.com/developerworks/cn/web/1003zhangjing_d2r/
[16] 曹玉平,龚主杰等. 关联数据技术及其研究现状[J]. 图书馆理论与实践, 2014.11: 42-45
[17] Chris Bizer, Richard Cyganiak, Tom Heath. How to publish linked data on the Web [EB/OL]. [2012-10-10]. http://www4. Wiwiss. fu-berlin. de/bizer/pub/LinkedDataTutorial/
[18] 沈志宏,刘筱敏等. 关联数据发布流程与关键问题研究——以科技文献、科学数据发布为例[J]. 中国图书馆学报, 2013.3: 53-60
[19] 夏翠娟,刘炜等. 关联数据发布技术及其实现——以 Drupal 为例[J]. 中国图书馆学报, 2012.1: 49-57
[20] 李楠,张学福. 基于关联数据的知识发现模型研究[J]. 图书馆学研究. 2013.1: 73-77

第 8 章　基于本体的知识推理

本体中蕴含了丰富的语义知识，对本体知识库进行推理，是以本体作为信息组织形式，比传统数据库更具智能化，通过推理可以获得本体中特定形式的知识集合以及运用本体中的知识来辅助解决涉及语义的应用。

使用 OWL 本体作为信息检索的载体，用户能够方便地在概念上描述信息需求，构造复杂的语义查询。本体的查询需要借助于 SPARQL 语言，但由于该语言只能查询 OWL 知识库中显式定义的知识，本身不具备推理的功能，因此需要借助推理引擎，将本体中具有隐含语义关联的数据提取出来，获得所有相关联的数据作为 SPARQL 查询的数据源。对本体中蕴含的知识进行有效地查询与智能化推理，是本体在各个领域能够成功应用的基础和关键。

本体的知识推理从根本上说就是把隐含在显式定义和声明中的知识通过一种处理机制提取出来。对本体的开发人员来说，本体的推理可以用于检测本体定义中存在的冲突，消除不一致性，优化本体表达和实现本体融合；而对于知识管理、语义检索、自然语言理解等诸多领域的本体使用者来说，本体的推理可以获得本体中特定形式的知识集合并用于解决实际问题。

8.1　本体推理机系统构成

本体推理机是实现语义检索和知识推理的关键技术之一。目前一些本体推理系统[1]已经成功用于推理和查询语义 Web，其中比较典型的有 W3C 对本体进行测试的本体推理机，DIG 基于描述逻辑实现的本体推理机，一些

集成在语义网开发平台(如 HP 实验室的 Jenal、德国 Karlsruhe 大学的 KAON2)和本体管理系统(如 IBM 的 SNOBASE 系统)中的推理引擎。

本体推理机由本体解析器、查询分析器、推理引擎、结果展现和 API 五大模块组成,如图 8-1 所示。

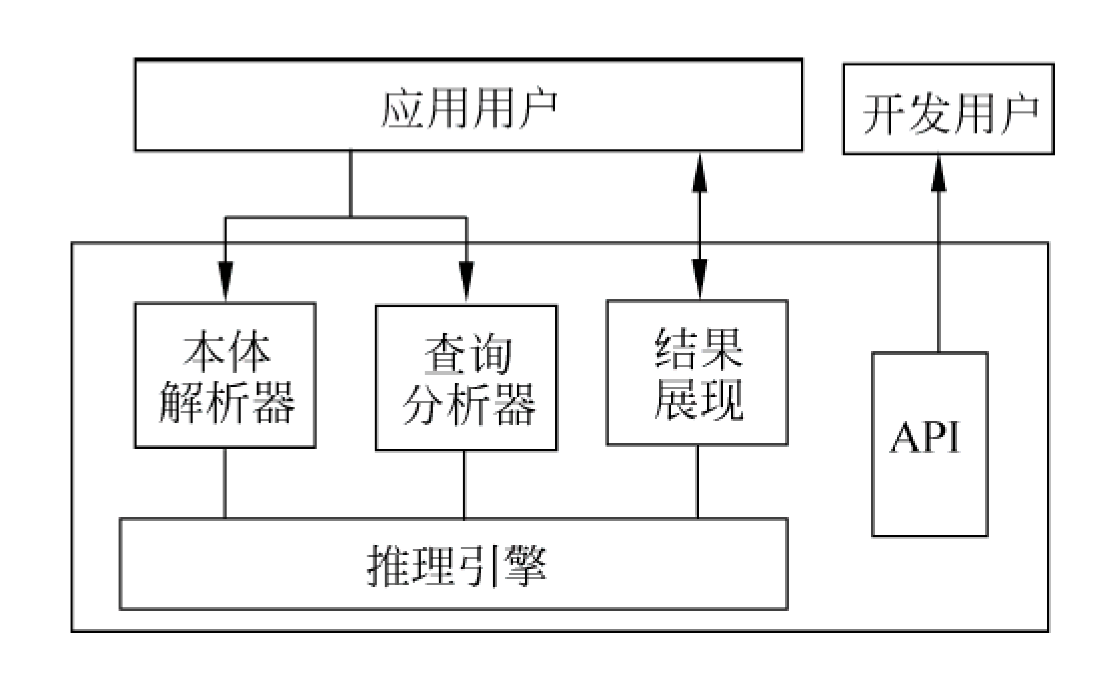


图 8-1　本体推理机的系统构成

本体解析器:负责读取和解析本体文件,它决定了推理机能够支持的本体文件格式。解析性能的好坏决定了推理机是否支持对大本体文件的解析。

查询解析器:负责解析用户的查询命令。

推理引擎:是本体推理机的核心部件,负责接受解析后的本体文件和查询命令,并执行推理流程,推理引擎决定本体推理机的推理能力。目前大部分推理引擎是基于描述逻辑的。

结果展现模块:对推理引擎所推导出来的结果进行包装,满足用户的需求。它决定了本体推理机能够支持的文件输出格式。

API 模块:主要面向开发用户,一般有三部分,OWL-API、DIG(DL Implementation GrouP)接口以及编程语言开发接口。OWL-API 为用户操作 OWL 本体文件提供了一种标准接口。DIG 接口为描述逻辑推理机系统向外提供服务提供了一组标准的接口,作用类似于数据库中的 ODBC,它允许前端(如本体编辑器 Protege)挂接到后台不同的推理引擎上。另外本体推理机提供的常见编程语言接口主要有 Lisp 和 Java 两种,大部分本体推理机是采用这两种编程语言实现的。

8.2　本体推理技术和推理算法

知识推理技术是基于本体知识库的基础，实现本体推理的主要技术[2]有以下四种：

1. 基于传统描述逻辑的推理方法

典型代表有 Pellet、Racer 和 FaCT++，它们都是基于传统 Tableaux 算法设计并实现的本体推理机，同时也引入了许多 Tableaux 算法的优化技术，从而使得推理效率很高。

2. 基于规则的方法

本体推理作为一类应用，可以映射到规则推理引擎上进行推理。目前已经存在了许多现成的实现 OWL 到规则的转化工具。基于规则方法实现的本体推理机系统典型代表有 Jess 和 Jena。

3. 利用逻辑编程方法

基于演绎数据库（Deductive database）技术实现，典型的系统项目有 F-OWL，德国卡尔斯鲁厄大学的 KAON2 也是一个采用这种技术方法实现的典型例子。

4. 基于一阶谓词证明器的方法

由于 OWL 声明语句能够很方便地转化为 FOL（First Order Logic），因此也就可以很方便地利用传统的一阶谓词证明器实现对 OWL 的推理，例如 Hoolet 本体推理机就是利用了 Vampire 一阶谓词证明器来实现本体推理。

用于语义推理的推理算法很多，其中最为常用的算法包括 Tableau 算法和 Rete 算法。目前各种流行的推理机所使用的算法多为这两种算法的优化算法，均以 Tableau 算法和 Rete 算法为基础。

1）Tableau 算法

Tableau 算法，是传统描述逻辑推理系统的核心算法，因为所有其他的推理功能如分类、包含等都可以规约为概念的一致性检查，而 Tableau 算法就是具体负责概念一致性检查，它最早由 Schmidt-Schau 和 Smolka 为检验 ALC 概念的可满足性而提出，该算法能在多项式时间内判断描述逻辑 ALC 概念的可满足性问题，Tableau 算法被广泛用于各种描述逻辑中以判断概念的可满足性或概念间的包含关系，各种优化的 Tableau 算法也已在实用推理机，例如 FaCT、Racer 等中得以实现。

Tableau 算法其基本思想是试图通过构建一个 C 的模型来证明概念 C 的满意度。一个满足 R 的推断中 CI 不为空。算法在一个树（或树集合）上运行，各节点被定制为 C 的子概念集，边被定制为 C 上发生的角色集。在树中的节点相当于在概念（角色）的推断中的元素（元组）。通常，一个单独的树从被定制为{C}的根节点初始化。

该算法竭尽全力应用 Tableau 规则分解节点上定制的概念的语法结构，或者扩展节点标志，向树中添加新的边和节点，或者合并边和节点。一个规则的应用有效地说明了从概念到应用的规则暗示了推断的约束，例如，如果 A B 是一个节点 x 的标志，那么规则被添加到 A 和 B 的标志中。这说明一个事实，那就是如果 x (A B)I，那么 x AI 且 x BI。简单的说，如果 R. A 是一个节点 x 的标志，那么规则添加一个标志为{A}的新的节点 y，它在 x 和 y 之间带有一个标志为{R}的边。这也说明一个事实：如果 x (R. A)I，那么就必须存在一个节点 y，使得（x，y）RI，y AI。

如果存在明显的矛盾，那么建立模型的尝试就失败了，这通常被称为产生了冲突。例如，如果包含某个概念 D 和 D 的某个节点，如果没有新的规则需要被应用了，而且一直没有冲突产生，那么建立的模型就成功了。当且仅当规则可以按照让模型被成功的构建的方式应用，证明一个概念是满意的就相对简单了。算法的计算复杂度由于一些规则不确定的事实而增长。事实上当冲突产生时，处理方式是通过回溯试图找到另一个不确定的规则进行应用。

这一基本理论用于 SHOQ (D_n) 中。首先，算法在一个森林或树上运行，需为概念 C 中每一个名词（术语）构建一个附加的树。被称为模块化的循环检测的形式也必须被用于确保终止。其次，算法需要使用一个类型检测器来检测源自数据类型存在性、值、最大概念和最小概念等的推断的约束。一般来说基于 Tableau 算法实现的本体推理机具有图 8-2 所示的典型系统结构。

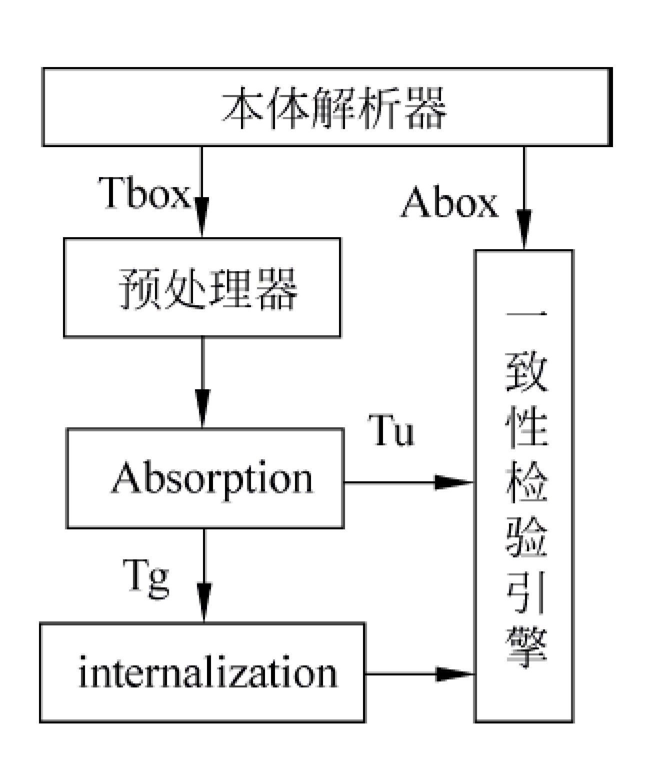


图 8-2　基于 Tableau 算法实现的推理机系统结构图

其中本体解析器负责解析本体和获取本体中 TBox 知识和 ABox 知识，并分别把它们送入预处理器和一致性检查引擎中；预处理器主要负责把概念表达式进行范式化，即把概念描述转换成一个标准的否定范式，然后再执行简化操作，经过预处理之后，包含和一致性问题能够得到简化，有时可以通过语法检测明显的可满足性，从而完全避免了比较耗时的一致性检查；Absorption 操作主要是为了把一般公理转化成原子定义公理，通过这个 Absorption 操作，能够大幅度提高推理机的性能，经过上述两步之后 TBox 分成了 Tu 和 Tg 两大部分，Tu 直接送入一致性检查引擎，Tg 则还需要通过一个内化操作，它可以把一个全局属性的问题归约为单个概念的可满足性检查，除了上述的体系结构图所描述的一些基本优化器之外，还有许多的优化器没有整入，譬如偏序优化，它在对 KB 进行分类的过程中可以减少包含测试的次数，还有包含检查优化，它可以避免采取代价昂贵的一致性检查策略，同样如果采用了一致性检查优化策略还可以进一步减少一致性检查时间。

2）Rete 算法

Rete 算法的发展是为了解决规则范式如何快速匹配事实这一核心问题。1982 年由美国卡耐基-梅隆大学的 Forgy 教授在《人工智能》杂志上提出，原理是将匹配过程中生成的中间结果一直存储在内存中。例如，在规则系统的常规执行中，传统做法是将事实与规则范式比较直至两个事实都被

断言，但在 Rete 算法中，事实只匹配一次，范式中的变量(x)就被存储起来。接着，如果另一个事实被断言，就只用简单的比较 x 的值，如果相等，这个规则就被激活了。

在基于规则的语言中，范式是组成规则的元素。Rete 算法决定规则的运行。Rete 是用一个静态判别网络，由语言编辑器生成，在规则间表示数据的信任。Rete 避免了在工作存储中进行数据添加期间进行不必要的重复匹配计算，取而代之的是在网络中被称为 beta 存储的节点中存储中间匹配变量。从(或向)工作存储添加(或删除)数据在 Rete 中试均衡的：删除数据的操作序列和添加数据是相同的。Rete 用空间换取时间：匹配的结果加入存储器中为以后重用做准备。

Rete 算法奇迹般的减少了匹配的数量，匹配的数量通过维持一个包含了内存中信息的有向图数据结构在每次循环中必须被执行。途中的第一层节点被称为 alpha 节点。每一个 alpha 节点表示一个单一的范式。当规则中不止一个范式时，alpha 节点被 beta 节点连接起来完成两个节点的联合。Beta 节点再连接到更远的 beta 节点，直至生成一个单独的节点。这个最终的节点表示了完整的规则。

当一个事实被断言时，一个令牌就被创建并被送入 Rete 网路中处理该事实类型的 alpha 节点。令牌是一个关于该事实并包含特殊信息的包，这些特殊信息比如该事实是被断言还是退回。如果这个令牌在一个 alpha 节点中匹配范式成功，它将被传到下一个节点，同时范式中所有的变量都被存入一张表中，这被称为节点存储。Beta 节点从 alpha 节点的存储表中获取行并试图将它们合在一起，验证同名的变量是否值也相等。例如，如果一张表中某行包含变量 A=1 和 B=2，另一张表中的一行包含变量 A=1 和 C=3，于是一个新行就被创建在 beta 节点中，这行包含值 A=1，B=2，C=3，于是令牌通过了。如果两张表中 A 的值不相同就不会生成新的行，令牌也不会被通过。当令牌在 Rete 网络中走过所有的路径，就会出现一个表示整个规则的终端节点。由于令牌到达终端节点，规则中的所有的范式必须已经被匹配，因此规则能被激活。Rete 算法的主要优势在于规则的条件在一个事实

被断言或删除时仅被再评估。用这种方式，断言一个新的事实仅仅是向网络中传递一个令牌，只有很小数量的匹配被执行。在常规执行那个，每一个新的事实都将与每一个规则的每个范式进行比较，这意味着大量的时间复杂度。撤销一个事实与断言是一样的，但是信息在节点存储中被清除了。

Rete 算法有编译时间和运行时间之分。在编译时间规则的前件被编译到一个判别网络中，用操作代码语言表示。Rete 网络是一个数据流网络，在规则环境下表示数据的相关性。在运行时间数据项表示工作存储中内容的变化，称为变换令牌，在网络的根部进入沿着路径执行。变换令牌或者表示向工作存储中添加操作或者表示从中删除操作。网络包含有两种节点：测试节点和联合节点。

Rete 算法的使用大大提高了规则匹配的效率，但是它为保存中间结果占用了大量的存储空间。由于存储空间的几何增长，以存储空间换取执行的时间，当推理的知识库比较大时，对存储空间的消耗是惊人的。为提高 Rete 算法的存储效率，减少存储空间的消耗，一些改进算法（如 TREAT，RETE *）也相应提了出来。

8.3　本体推理机分类

根据其使用方式的不同可以把本体推理机分为内置和外挂两大类，这种区分主要针对本体推理机的具体使用方式而定，如 SWOOP 编辑器内置了 Pellet 本体推理机，则此时的 Pellet 对用户来说是内置的，但是如果 Pellet 通过 DIG 接口与 Protege 本体编辑器连接在一起之后，此时 Pellet 对用户来说就是外挂推理机；根据推理机实现的技术不同而分成基于描述逻辑、基于规则等不同类别，这里是按照本体推理机是否针对某些具体本体描述语言而实现把本体推理机划分为专用和通用两大类：

（1）专用本体推理机。如 Racer、FaCT++、Pellet 等属于专用本体推理机，它们支持的主要本体语言为 RDFS、OWL 等。专用本体推理机效率较高，使用方便，只是它将推理能力限定在几种具体的本体语言上，较难进行

扩展。

(2) 通用本体推理机。典型的有 Jess,它是开放的,用户只需要提供不同领域的推理规则 Jess 就可以对不同领域进行推理。通用本体推理机效率低,也不能提供针对各种具体领域的优化能力,使得这种推理机的效率很难被优化。

两类推理机都应实现两个基本推理功能:

(1) 检查本体的一致性。保证本体一致性就是保证本体中已获得的类和个体逻辑上的一致性,检验实例是否与类、属性和个体的所有公理约束相冲突。

(2) 得到隐含的知识。本体创建一般遵循在尽量简化本体的同时使得本体尽量包含足够多的信息。因为如果要在一个本体中声明出所有的语义关系,那么构建本体将是一件非常复杂而又繁琐的任务,也会导致本体过于庞大而难以处理;若本体设计简单,在实际应用中又需要本体中蕴含的语义信息,这时就需要本体推理机来获取本体中隐含的信息。

8.4 典型的本体推理机系统

表 8-1 为当前在语义推理方面使用最为流行的推理机列表。

表 8-1 最为流行的推理机

名称	开发组织	URL	开源
JENA	HP 实验室	http://jena. soursefouge. net/	开源
PELLET	美国马里兰大学 MINDSWAP 项目组	http://clarkparsia. com/pellet	开源
RACER	德国 Racer Systems	http://www. racer-syatem. com/	商用
JESS	美国 Sandia 实验室	http://www. jessrules. com/	开源
KANO2	德国卡尔斯鲁厄大学	http://kano2. semanticweb. org/	开源
FACT++	英国曼城斯特大学	http://owl. manac. uk/factplusplus/	开源

(1) Jena 是由 HP Labs(http://www. hpl. hp. corn)开发的(Java 开发)一种产生式规则的前向推理系统。Jena 框架包含一个本体子系统

(Ontology Subsystem)，它提供的 API 支持基于 OWL，DAML＋OIL 和 RDFS 的本体数据；Jena 提供了 ARQ 查询引擎，实现 SPARQL 查询语言和 RDQL，从而支持对模型的查询。Jena 的主要特点是代码开源性。

(2) Pellet 是由美国马里兰大学 College Park 分校 MinSwap 实验室开发的一个本体推理机。是一个基于 Java 的开放源码系统，以描述逻辑作为理论基础，采用 Tableaux 算法。Pellet 是一个较完善的 OWL-DL 推理机，广泛支持个体推理，包括名义(nominal 枚举类)推理和合取查询，用户自定义数据类型和本体的调试支持。Pellet 主要应用在本体开发、发现和构建 Web Service 等方面。Pellet 效率较高，但是缺乏对本体规则语言 SWRL 的支持，并且支持的本体查询语言不够全面。一般只是进行 A-Box 推理查询的时候考虑使用 Pellet，而查询如果牵涉到 T-Box 推理则推荐使用 Racer。

(3) Racer 最初由德国的汉堡大学开发的基于描述逻辑系统的知识表达系统，采用 Tableaux 算法，它的核心系统是 SHIQ(描述逻辑的一种，它主要包含交、并、存在、任意、数量约束等构造算子)。Racer 也可以对基于 RDFS\OIL＋DAML 和 OWL 知识库进行处理。它提供支持多个 TBox(术语公理)和 ABox(断言事实)的推理功能。给定一个 TBox 后，Racer 可以完成各种查询服务。Racer 具有较强的本体一致性检查功能，在 TBox 方面推理能力较强，能够对大本体文件提供良好的支持，而且具有图形用户界面，并有详细的开发文档和示例代码，但是 Racer 不支持对枚举类和用户自定义数据类型的推理。

(4) Jess 是一个经过扩充的 CLIPS(C Language Integrated Production System)版本，由美国实验室分布式系统计算机组用 Java 实现的基于产生式的前向推理引擎。Jess 是性能良好的开放式推理机，采用经典的 Rete 算法，支持正向和逆向推理，在提供这个领域的相关规则和事实信息的前提下，原则上可以处理各种领域的推理服务。由于 Jess 是通用推理引擎，使得这种推理机制的效率很难优化。

(5) KAON2 是管理 OWL-DL、AWRL 和 F-Logic 本体的基础设施，它由信息技术研究中心(FZI)的 IPE、卡尔斯鲁厄大学的 AIFB 和曼切斯特大

学的 IMG 共同努力开发的。KAON2 是 KAON(也即 KAON1)的新一代产品,KAON2 与 KAON1 的不同在于,KAON1 基于 RDFS 及其扩展,而 KAON2 基于 OWL-DL 和 F-Logic。KAON2 的特点主要体现在以下几个方面:

① 提供 OWL-DL, SWRL 和 F-logic 本体的 API;

② 利用 RMI 以分布式方式提供对本体的访问的独立服务器;

③ 回答以 SPARQL 表示的连接查询的推理引擎;

④ DIG 的界面,允许从 PROTÉGÉ 等工具访问;

⑤ 抽取关系型数据库中现有的本体实体的模块;

⑥ KAON2 的 API 能够处理 OWL-DL 本体,API 可以读取 OWL XML Presentation Syntax 和 OWL RDF Synatx;

⑦ 对于推理,KAON2 支持 SHIQ(D). 这是 OWL-DL 的子集。这包括除 nominals 以外的 OWL-DL 的特征,因为 nominals 不是 OWL Lite 的组成部分,KAON2 支持所有的 OWL Lite;

⑧ KAON2 也支持 SWRL 的所谓的 DL-safe 的子集。这个约束可以使得推理可决定;

⑨ KAON2 的 API 也能处理 F-logic 的本体,它支持 F-Logic 的功能子集;

⑩ KAON2 中查询可以用 SPARQL 构造,但是并不支持所有的 SPARQL 规范。

采用 KAON2 的项目包括 EU IST OntoGov, EUISTSEKT 项目, BMBF 的 SmartWeb 项目,但是,近几年 KAON2 很少进行进一步的开发以及维护工作。

(6) FaCT++是 FaCT(Fast classification of Terminologies)的新一代产品,是英国曼城斯特大学开发的一个描述逻辑分类器,提供对模型逻辑(model logic)的可满足性测试,采用了客户端/服务器模式。FaCT++采用 FaCT 的算法,二者均采用 tableaux 算法,但是内部结构是不同的。FaCT++是基于描述逻辑的推理机,支持 OWL DL 和 OWL2 标准,为了提高

效率和获得更好的平台移植性，FaCT++采用了 C++而非 FaCT 的 Lisp 语言来实现，这是 FaCT++和其他推理机不同的一个地方。FaCT++推理机在本体一致性检查上具有很好的表现，但是 FaCT++没有提供 OWL 接口也不支持对实例的查询。FaCT++也是开源的，但是开发文档和示例代码都不详细，而且没有友好的用户界面。

通过对当前本体推理机的分析和几种典型推理机的测试对比，发现尽管大部分本体推理机都能够实现两大基本推理功能，但是还是存在着一些不足，如 Racer 不支持对枚举类和用户自定义数据类型的推理，Pellet 缺乏对本体规则语言 SWRL 的支持并且支持的本体查询语言不够全面等。另外，它们还缺乏对多个本体、不一致本体以及大规模本体服务器的推理支持，还有用户界面不够友好等缺陷。

由此可见，未来本体推理机的发展趋势应该是在系统功能方面开发更为强大和完善的推理算法，如在描述逻辑中支持量词约束、用户自定义数据类型和逆关系属性类型等。并能允许用户自定义推理规则以及支持多个、多版本和不一致本体的推理。向用户提供更加友好的 GUI 和更为丰富的程序开发接口也将是未来本体推理机的一大发展趋势。

8.5　粗逻辑在本体推理中的应用

描述逻辑的知识描述能力和推理能力受到描述逻辑算子个数的限制，若需要描述逻辑完成更多的推理任务，就需要增加描述逻辑的算子。本节的目的在于应用粗糙集理论来处理不确定集合，从而使得相应的本体能够有效地处理不确定信息，将粗逻辑引入描述逻辑的知识表示和推理机制。

8.5.1　描述逻辑

描述逻辑（Description Logic，DL）又称为术语逻辑（Terminology Logic）或类 KL-ONE 系统，由 Brachman 于 1977 年在他的博士论文中首先提出，并实现第一个 DL 系统 KL-ONE，最初的研究动机是为知识表示中的

语义网络(Semantic Network)提供形式化基础。描述逻辑的基本构件是概念(Concept)、角色(Role)和个体(Individuals),简单的概念和角色可以通过复合方式表示复杂的概念和角色。概念描述了一个个体集合的共同属性,并且可将概念解释为对象集的一元谓词,将关系解释为对象之间的二元关系。DLS(Description Logic System)将推理作为中心服务,即从知识库以显式包含的知识推导出隐含表示的知识。

描述逻辑系统的发展大致可分为四个阶段。从 1980 年至 1990 年,称为前描述逻辑系统(Pre DL Systems),主要关注对一些系统的实现,比如 KL-ONE、K-REP、BACK 和 LOOM 等,这些系统都基于结构化(Structural)的包含关系算法;第二个阶段从 1990 年到 1995 年,这一阶段开发了基于表(Tableau)的算法,并实现了第一批以该算法为基础的系统。从这个阶段开始,人们正是开始对各种 DLS 的推理复杂性进行分析。另外一个重要的发现是描述逻辑和模态逻辑(Modal Logic)有着密切的联系;第三个阶段从 1995 年到 2000 年。这个阶段开发了许多优化的 Tableau 算法,并且优化的逻辑描述系统(FACT、RACE 和 DLP)证明了这些优化 Tableau 算法的优良性能;第四个阶段从 2000 年至今,在这个阶段开发了具有商业价值的 DL 系统,其中采用了较强的描述语言以及基于 Tableau 的算法。描述逻辑在计算机领域开始受到重视并进行推广。

DL 最基本的描述语言是 AL(Attribute Language),在 AL 定义的仅仅是原子否定。ALC(Attribute Language Complements)是在 AL 语言上加上对任意一个概念的否定。如果使用 A 和 B 来表示原子概念,R 表示原子关系,C 和 D 表示概念描述,I 是一个解释,则可以用表 8-2 来说明 ALC 的语法和语义。

表 8-2 ALC 语义及语法

构造算子	语法	语义	例子
Atomic concept	A	$A^I \in \Delta^I$	Human
Top concept	T	Δ^I	Male ∪ ¬ Male
Bottom concept	⊥	Φ	Male ∩ ¬ Male
Atomic negation	$\neg A$	$\Delta^I \backslash A^I$	¬ Male

续表

构造算子	语法	语　义	例　子
Concept negation	$\neg C$	$\Delta^I \backslash C^I$	
Concept disjunction	$C \cup D$	$C^I \cup D^I$	Man $\cup$ Woman
Concept conjunction	$C \cap D$	$C^I \cap D^I$	Human $\cap$ Male
Existential	$\exists R.C$	$\{a \in \Delta^I \mid \exists b.(a,b) \in R^I \wedge b \in C^I\}$	$\exists$ has-Child. Male
Value restriction	$\forall R.C$	$\{a \in \Delta^I \mid \forall b.(a,b) \in R^I \rightarrow b \in C^I\}$	$\forall$ has-Child. Doctor

在描述逻辑语义中，语义是通过映射$(\Delta^I, {}^I)$来定义的，其中 Δ^I 为论域。映射函数 I 把一个概念映射为 Δ^I 的一个子集，把关系映射为 $\Delta^I \times \Delta^I$ 的一个子集。例如概念 A 的语义为：$A^I \in \Delta^I$，关系 R 的语义为 $R \in \Delta^I \times \Delta^I$。

描述逻辑知识表示系统包含了一个知识库(Knowledge base)及其推理服务(Reasoning)。其中知识库包含两个组成部分：术语集或术语知识库 Tbox(Terminology axiom)和断言集或断言知识库 Abox(Assertions axiom)，它们之间的组成关系如图 8-3 所示。Tbox 是相关概念和关系的术语公理集合，专门用来描述应用领域中内涵概念和关系的一般属性；Abox 是相关个体的实例化断言的集合，表示应用领域外延知识中个体的关系。

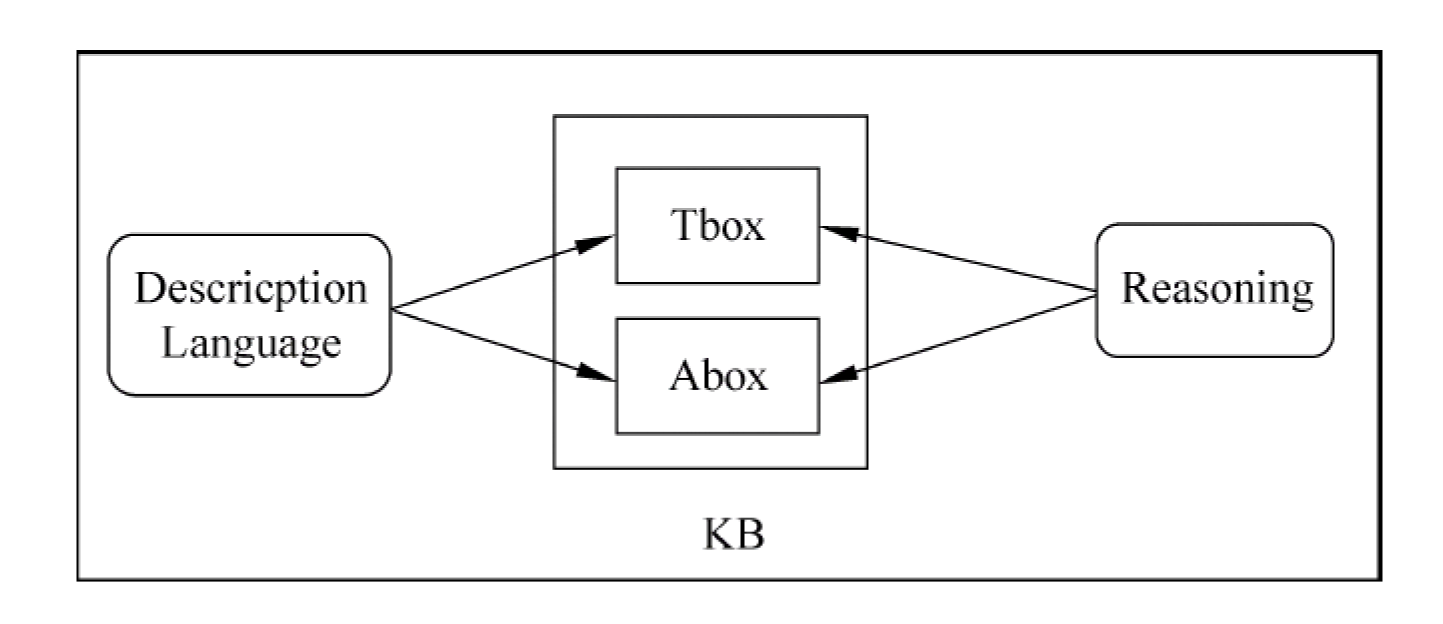


图 8-3　描述逻辑的体系结构

1. TBox：Tbox 术语公理包括两种形式

假设 C、D 表示概念，R、S 表示关系

(1) 蕴含公理(Inclusion)：$C \vee D(R \vee S)$。这个公理定义了包含关系，可实现概念分类。

(2) 等价公理(Equality):$C'D(R'S)$。这个公理用来定义领域中复杂的概念。

公式的语义是根据期望定义的,如果解释 I 满足 $C^I \subseteq D^I$,那么 $C \vee D$,同时如果解释 I 能够使 $C^I = D^I$,那么 $C'D$。令 T 是一个公式的集合,如果解释 I 对 T 中的每一个公式都满足,即 I 满足 T;若解释 I 满足一个公式(或者一个公式的集合),那么解释 I 是这个公式(或公式集合)的一个模型;如果两个公式(或者公式集合)具有同样的模型,那么将他们叫做等价的。

2. ABox:ABox 中的实例化公理主要有以下的几种形式构成

假设 a、b 表示个体,C 表示概念,R 表示关系

(1) 概念断言 $C(a)$:表示个体 a 属于概念 C。

(2) 关系断言 $R(a,b)$:个体 a 与 b 存在着关系 R。

对于解释 I,如果 $a^I \in C^I$ 或者 $(a^I, b^I) \in R^I$,那么称解释 I 满足断言 $C(a)$ 或者 $R(a,b)$。若解释 I 满足 ABox 中的 A 的每个断言,即解释 I 满足 A,并称解释 I 是 ABox 的一个模型。

8.5.2 描述逻辑的推理机制

描述逻辑推理机的作用是为知识库(KB)提供推理服务,使其能够发现在表面信息内部的隐含信息,对于知识发现、维护和系统改进是十分有利的。

在 Tbox 中的推理问题,主要包括四个方面,令概念 C 和 D 是 T 中的两个概念:

(1) 可满足性检测(Satisfiabilty):如果存在 T 中的一个模型 I,使得 C^I 是非空的,从而根据 T,概念 C 是满足的,同时也可以说 I 是 C 的模型;

(2) 包含关系检测(Subsumption):如果对于 T 中的每个模型 I,都有 $C^I \subseteq D^I$,那么根据 T 概念 C 被概念 D 包含,可以写成 $C \vee_T D$,也可以称作 $T\ \ C \vee D$;

(3) 等价关系检测(Equivalence):如果对于 T 的每个模型 I,都使得

$C^I = D^I$,那么根据 T,概念 C 和 D 是等价的,被写作 $C'_T D$,或者 $T\ \ C'D$;

（4）相离关系检测(Disjointness):如果对于 T 的每个模型 I,都有 $C^I \cap D^I = \varnothing$,那么依据 T,概念 C 与概念 D 是相离的。

与 Tbox 相对应,ABox 中的推理中最基本的是一致性检测。

一致性检测即如果一个由 Tbox 中的 T 扩展的 ABox 中的 A 是一致的,那么必存在一个 A 和 T 共有的模型 I。

ABox 的推理还包括实例检测,即对给定的个体和概念进行匹配,检验它们之间是否存在实例关系;实现(Realization),即找到给定的个体的最基本的概念;返回(retrieval),即个体的获取,是指在知识库中找到一个给定的概念的所有实例。

基于描述逻辑 ALC 的推理机制,对于知识库 KB 的本体推理,TBox 的四个推理任务:可满足性检测、包含关系检测、等价关系检测和相离关系检测之间存在如下的转换关系:

（1）C 是不可满足的$\Leftrightarrow C$ 被? 包含;

（2）C 和 D 是等价的$\Leftrightarrow C$ 被 D 包含,同时 D 被 C 包含;

（3）C 和 D 相离$\Leftrightarrow C \cap D$ 被? 包含;

（4）C 被 D 包含$\Leftrightarrow C \cap \neg D$ 不可满足;

（5）C 和 D 是等价的$\Leftrightarrow C \cap \neg D$ 与 $D \cap \neg C$ 同时不满足;

（6）C 和 D 是相离的$\Leftrightarrow C \cap D$ 是不可满足的。

即在 TBox 层面上的推理问题都可以转换为包含性推理或者可满足性定理。ABox 主要的推理任务是一致性检测和实例检测,其余的概念实现,个体获取等推理任务都可以转化为一致性检验,而一致性检测同时又能够与 TBox 中的不可满足性检测相互转换,这样就需要一个出色的算法来处理不可满足性判定来完成一致性检测,即 Tableau 算法。

8.5.3 粗逻辑

粗糙集(Rough Set)是由 Pawlak Z. 在 1982 年提出的,其基本思想是通过案例库分类归纳出来的概念和规则。粗糙集的基本概念是上近似和下近

似，使其对于不能用已有概念表示的新概念给出一个近似的表示。经典逻辑对应着经典集合，粗逻辑也是对应着粗糙集而产生的，即使用粗糙集来描述可能世界。

1. 粗糙集

在介绍粗糙集之前首先要给出近似空间的概念，近似空间是有一个二元组 $A=(U,R)$ 组成，其中 U 是一个依赖与专业领域对象的集合，R 是一个划分方案，R 可以将 U 划分为若干个子集 E_i，$i=1,2,\cdots,n$，我们称 E_i 为等价类，是商集 U/R 的元素，其中 $U/R=\{E_1,E_2,\cdots,E_n\}$，其中空集 $\varnothing$ 和 E_i 是基本集，一个或者多个基本集被并称之为合成集，我们把全体合成集的族称之为 $com(A)$。在近似空间 A 中对于任意的集合 $X\subseteq U$ 都可以定义它的上近似集和下近似集：

$$\overline{R}(X)=\bigcup_{E_I\cap X\neq\varnothing}E_i=\{y\mid(\exists y)(y\in[x]_R\wedge y\in X)\}\qquad(8\text{-}1)$$

$$\underline{R}(X)=\bigcup_{E_I\subseteq X}E_i=\{y\mid(\forall y)(y\in[x]_R\rightarrow y\in X)\}\qquad(8\text{-}2)$$

其中，$[x]_R$ 是包含 x 的等价类。下近似集 $\underline{R}(X)$ 是那些 X 的子集的基本集的并集，因此这个下近似集是 X 所包含的最大合成集；上近似集 $\overline{R}(X)$ 是那些所有与 X 有交集元素的基本集的并集，因此这个上近似集是包含 X 的最小合成集。我们称 $\underline{R}(X)\subseteq X\subseteq\overline{R}(X)$ 为在近似空间 A 的粗糙集。同时还有：$BN_R(X)=\overline{R}(X)-\underline{R}(X)$ 是 X 在 A 上的界集。$\sim X=U-X$ 是 X 关于全集 U 的补集。

2. 粗逻辑

以近似空间 A 为基础的粗逻辑，又被称为 R-逻辑。和描述逻辑类似，它用 $p,q,r,\cdots$ 来表示命题符号；用？和 T 来分别表示底层概念和上层概念；它的基本连接词是 $\sim,\wedge,\vee$，相应的可以和描述逻辑中的算子：、u、t 等价；同样使用小写字母 $\varphi,\phi,\cdots$ 表示逻辑公式。解释 $I:\Gamma\rightarrow 2^U$，其中 Γ 是 R-逻辑公式集合，2^U 是 U 上全体子集的集合，即 Γ 中的每个公式都可以被解

释为 U 中的子集，

$$\forall \varphi \in \Gamma, \quad \varphi^I = \{\omega \subseteq U \mid u_\omega(\varphi) = T\}$$

其中 u_ω 是子集 ω 上的赋值函数。于是得到 R-逻辑公式与近似空间 A 上的子集存在如下的对应关系：

(1) $T^I = 2^U$；

(2) $\bot^I = \phi$；

(3) $(\sim\varphi)^I = \sim\varphi^I$；

(4) $(\varphi \vee \phi)^I = \phi^I \cup \varphi^I$；

(5) $(\varphi \wedge \phi)^I = \phi^I \cap \varphi^I$；

(6) $(\varphi \rightarrow \phi)^I = \sim\varphi^I \cup \phi^I$；

(7) $(\varphi \leftrightarrow \phi)^I = (\sim\varphi^I \cup \phi^I) \cap (\sim\phi^I \cup \varphi^I)$。

同时分别称 $L = \underline{R}$ 和 $H = \overline{R}$ 是粗糙下似算子和粗糙上近似算子，这两个算子都是一元操作，因此它可以被用来作用于集合和公式两种意义，分别使用括号以示区别，即：若它是作用于集合的，则需要对其加括号；若作用于非集合，则不需要加括号。从而有：

(8) $(L\varphi)^I = L(\varphi) = \underline{R}(\varphi^I)$；

(9) $(H\varphi)^I = H(\varphi) = \overline{R}(\varphi^I)$。

3. 粗逻辑的近似精度(Degree of True)

在 R-逻辑中，我们需要引入近似精度的概念。我们令 $X \subseteq U$ 是近似空间 $A = (U, R)$ 上的粗糙集，$K(S)$ 表示集合 S 中元素的个数或基数，则

$$\eta_A(X) = K(\underline{R}(X)) / K(\overline{R}(X)) \qquad (8\text{-}3)$$

是 X 在 A 上的近似精度，简记为 $\eta_A(X)$ 为 η。

显然有 $0 \leqslant \eta \leqslant 1$，由于 $\underline{R}(X) \subseteq X \subseteq \overline{R}(X)$，这样当 $u_\omega(\varphi) = T$ 时，则称 φ 在可能世界 $\omega \in 2^U$ 上取真值；当 $u_\omega(\varphi) = \bot$ 时，则称 φ 在可能世界 $\omega \in 2^U$ 上取假值；若 $u_\omega(\varphi) = \eta$，则称 φ 在可能世界 $\omega \in 2^U$ 上的近似精度值，即不完全真。

若 $u_\omega(\varphi) = \eta \geqslant 0.5$ 时，称 φ 在 $\omega \in 2^U$ 上是可接受的精度值，即粗糙真；

反之，其在 $\omega \in 2^U$ 上是不可接受的精度值或者说是粗糙假。

8.5.4 粗逻辑在描述逻辑推理中的应用

在本节将要探讨粗逻辑在描述逻辑中的应用[3]，在现实世界中，很多概念的界限并不是如同表 8-1 所举的例子（Man′Male∪Female）中提到的具有明确的分类，同时即使在分类明确的基础上还存在一系列无法明确表述的例子，例如，我们知道 KindMan 是 Man 的一个子类，但是在 Man 被 Male 与 Female 划分的情况下，KindMan 是无法被描述的。

为了解决描述逻辑处理不确定信息的问题，有学者提出了把模糊逻辑与描述逻辑相结合，形成了模糊描述逻辑。但是基于模糊数学的模糊逻辑具有天生的缺陷，即隶属度的人为因素过高，因此本节尝试使用粗逻辑参与描述逻辑的知识表示与推理，给出了一个初步的框架。

1. 粗逻辑参与的 TBox

通过前文，我们已经知道描述逻辑的 TBox 部分包括蕴含公理和等价公理，即若 C、D 表示概念，R、S 表示关系，I 表示对于概念的解释，显然当 C、D 为粗糙集的时候依然可以存在 $C \vee D(R \vee S)$，同时有 $C^I \subseteq D^I$，但是两个粗糙概念的等价关系就不会是那么简单就可以直接定义的，下面着重讨论当 C、D 表示粗糙概念的时候，两个概念的等价如何确定。

首先要明确一个事实，若 C、D 两个粗糙概念是完全相同的（可以说 D 是 C 的完全复制），那么必然对于任意的解释 I 有 $\overline{R}(C^I)=\overline{R}(D^I)$，$\underline{R}(C^I)=\underline{R}(D^I)$，即两个概念解释的上下相近似集相同；反之并不一定成立，但是在当前的近似精度下有无法找出两个概念的区别。

若举出一个极端的例子，两个概念任意解释的上近似是全集 U，下近似集为空集 $\varnothing$，可见在这个标准之下，全集 U 的所有子集都可以被认为是等价的，这个结论当然是错误的，因此确定在 TBox 下两个概念的等价关系的确认依赖于这两个粗糙概念的近似精度 η，于是有如下的定理：

定理 8.1 在描述逻辑 TBox 中，两个粗糙概念 C、D，它们若有分别相

同上下近似集的等价程度等于这两个概念的近似精度 η,即:

$$Equ(C,D)=K(\underline{R}(C))/K(\overline{R}(D)) \tag{8-4}$$

其中 $K(S)$表示集合 S 中元素的个数或基数,当然在 $Equ(C,D)=1$ 的时候,C、D 是普通集合,同时也可以确认 C、D 是完全等价的,这也与普通集合是粗糙集的特殊情况这个事实相吻合。

于是,我们可以得出两个概念 C、D 是否等价的算法:

(1) 计算 C、D 的上下近似集;

(2) 若 C、D 上下近似集分别相同,进入步骤 3,反之确认两者不等价;

(3) 使用公式(8-3)计算近似精度;

(4) 若近似度大于认定的阈值(一般设定为 0.5),可认定 C、D 等价,反之两者不等价。

2. 粗逻辑参与的 ABox

同样在前文中,ABox 包括了实例化公理,包括概念断言和关系断言,关于关系断言,描述逻辑与粗逻辑参与的描述逻辑并没有很大的差别,下面将重点讲述概念断言。

若 C 是一个粗糙概念,那么在现有划分下,只能通过上下近似集来描述这个概念,那么一个实例是否属于这个概念需要分两个方面来考虑:

(1) 若存在一个实例 a,对于粗糙概念 C,有 $a\in\underline{R}(C)$,则必然有永真断言 $C(a)$;

(2) 若存在一个实例 a,对于粗糙概念 C,有 $a\in BN(C)$,其中 $BN(C)=\overline{R}(C)-\underline{R}(C)$是 C 的边界,在这种情况下,就无法简单地用真与假来描述实例断言 $C(a)$。同样在前文的例子中,假设这个概念任意解释的上近似是全集 U,下近似集为空集$\varnothing$,那么所有实例都属于这个概念,这个结论当然也是无法接受的。因此这个实例 a 属于概念 C 的程度适合 C 的边界是密切相关的。

定理 8.2　对于 ABox 中的实例 a,$a\in BN(C)$,那么 a 属于概念 C 的属于度等价于粗糙概念 C 的近似精确度,即

$$Mem(a,C) = K(\underline{R}(C))/K(\overline{R}(C)) \tag{8-5}$$

这样就可以得出一个实例 a 是否为粗糙概念 C 的实例的算法：

(1) 若 a 属于 C 的下近似集，可以断定 a 属于 C，若 a 属于 C 的边界，进入步骤(2)，其余情况视为 a 不属于 C；

(2) 计算 C 的近似精确度 η；

(3) 若 η 大于给定阈值，视为 a 属于 C，反之认定 a 不属于 C。

3. 粗逻辑参与的推理机

在粗逻辑改造的 TBox 和 ABox 的基础之上，可以使用粗逻辑来参与描述逻辑的基本推理。前文中提到过描述逻辑的推理部分需要完成两个部分的任务：TBox 的可满足性检测、包含关系检测、等价关系检测和相离关系检测；ABox 的一致性检测。

对于 TBox 的可满足性检测、包含关系检测和等价关系检测三个任务，本文不再赘述，尤其是对等价关系检测，可以应用 4.3.1 小节所提供的算法来判断。

本文着重探讨相离关系的检测[4]，对于两个粗糙概念 C、D 的上近似集不相交，则必然有 C 与 D 的相离关系；若 C 与 D 的边界有交集那么就要考虑两个粗糙概念的近似精确度，两个粗糙概念的精确度越低，那么两个粗糙概念相交部分占两个粗糙概念实际的比重越低，即两个粗糙概念的相离度随着两个概念的近似精确度的增加而减少，于是有如下的定理：

定理 8.3 两个粗糙概念 C、D 的相离度与 C、D 的近似精度之和成反比，即

$$Dis(C,D) = \frac{K(\overline{R}(C) \cap \overline{R}(D))}{\eta(C) + \eta(D)} \tag{8-6}$$

当 C、D 都是普通集合的时候，$Dis(C,D)$的相离度达到最小但不会为零，除非分子为零，即两者的交集为空，这也是与事实相符的。

这样就能够得到相离关系检测的算法：

(1) 若 C、D 的上近似集无交集，视 C、D 相离的。若 C、D 的上近似集存在交集，转入步骤(2)；

(2) 计算 $Dis(C,D)$;

(3) 若 $Dis(C,D)$大于给定阈值,视 C、D 相离,反之视 C、D 不存在相离关系。

8.6　基于 Jena 的本体推理机

Jena 被设计成一个具有三层(Layer)架构和多种视图(View)的开发框架(Framework),它有多种应用程序开发接口提供给系统级和应用级的开发人员,具有很高的灵活性。具体包括用于对 RDF 文件和模型进行处理的 RDF API,用于对 RDF、RDFS、OWL 文件(基于 XML 语法)进行解析的解析器,RDF 模型的持续性存储方案,用于检索过程推理的基于规则的推理机子系统[5],用于对 Ontology 进行处理和操作的 Ontology 子系统,用于信息搜索的 RDQL 查询语言。

8.6.1　内置推理机

推理功能是 Jena 中的一个子系统,目的在于将推理机制与推理器引入到 Jena 系统中。Jena 提供的是基于规则的推理机,例如 RDF 推理机、OWL 推理机等,除了可以通过这些推理机进行一般推理外,用户还可以自定义推理规则,或者是引入其他推理引擎,如 Racer、Pellet 等。考虑到不同层次的推理需求,Jean 语义 Web 编程框架支持 4 种内置的推理机,在实际应用中,应根据推理的需求选用适当的推理模式。下面以文献[1]给出的 OWL 样例本体为例说明。

本体描述了 Mammal、Human 和 Canine 等概念及其关系,另外还建立了相应的个体。同时为了便于阅读,采用 Turtle 语法,基于三元组的形式对此 OWL 本体进行序列化。每一个三元组用谓词将不同的概念联系起来,同一个概念可以通过多个谓词与其他多个概念相联系。用 Turtle 语法表示的语句具有形式“subject $predicate_1$ $object_1$; $predicate_2$ $object_2$; …$predicate_n$… $object_n$”。

OWL 样例本体的代码如下：

```
#以下定义命名空间前缀
@ prefix ex:<http://example.org#>.
@ prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax.ns#>.
@prefix rdfs:<http://www.w3.0rg/2000/01/rdf-schema#>.
@ prefix owl:<http://www.w3.org/2002/07/owl#>.
#以下定义了 7 个类(概念)
ex: Mammal rdf: type owl: Class.
ex: Human rdf: type owl: Class; rdfs: subClassOf ex: Mammal.
ex: Canine rdf: type owl: Class; rdfs: subClassOf ex: Mammal;
owl: equivalentClass[rdf: type owl: Restriction; owl: onProperty ex: breed;
owl: someValuesFrom ex: Breed].
ex: PetOfRyan rdf: type owl: Class; owl: intersectionOf
(ex: Mammal[rdf: type Owl: Restriction; owl: onProperty ex: hasOwner;
owl: hasValue ex: Ryan]).
ex: Breed rdf: type owl: Class.
ex: LargeBreed rdf: type owl: Class: rdfs: subClassOf ex: Breed.
ex: SmallBreed rdf: type owl: Class; rdfs: subClassOf ex: Breed.
#以下定义了 2 个数据类型属性
ex: name rdf: type owl: DatatypeProperty.
ex: registeredName rdf: type owl: DatatypeProperty: rdfs: subPropertyOf ex: name.
#以下定义了 3 个对象属性
ex: breed rdf: type owl: ObjectProperty.
ex: hasOwner rdf: type owl: ObjectProperty.
ex: owns rdf: type owl: ObjeetProperty; owl: inverseOf ex: hasOwner.
#以上部分构成了 OWL 知识库的 TBOX
#以下定义了 5 个实例(个体),构成了知识库的 ABOX
ex: GoldenRetriever rdf: type ex: LargeBreed.
ex: Chihuahua rdf: type ex: SmallBreed.
ex: Ryan rdf: type ex: Human; ex∷name"Ryan Blace"; ex: owns ex: Daisy.
ex: Daisy rdf: type ex: Canine; ex: name"Daisy"; ex: breed ex: GoldenRetriever;
ex: registeredName"Morning Daisy Bathered in Sunshine".
ex: Amber rdf: type ex: Mammal; ex: name "Amber"; ex: breed ex: GoldenRetriever.
```

(1) RDFS 推理机：支持 RDFS 蕴含，实现基于 RDFS 中类和属性的关系进行推理。对样例 OWL 本体采用 RDFS 推理机进行推理，可以得到如

表 8-3 所示的关于个体的隐含信息。

(2) 传递推理机：仅支持 RDFS 中对称属性和传递属性的蕴含。如可以对家族本体中的 hasSibling 属性进行这种类型的推理。

(3) OWL 推理机：支持 OWL 蕴含的推理，涉及到的 OWL 构造包括 sameAs、SymmetricProperty 和 maxCardinality 等。目前，Jena 对 OWL 推理的支持还不够完备。对示例本体数据采用 OWL 推理机进行推理，可以推出如表 8-4 所示的关于个体的隐含信息。

(4) 通用规则引擎：该引擎包含了其自身的规则语言和实用的方法，可以构建出 if/then 形式的规则。一般将规则保存在程序中的字符串或项目的 rules 文件中，规则的格式如下：

[规则名：(三元组$_1$)，(三元组$_2$)，…(三元组$_n$) → (三元组$_{n+1}$)]

其中，前 n 个三元组为规则的前提，第 n+1 个三元组为规则的结论，每个三元组具有形式：subject predicate object。其中 subject 和 object 分别为主语和宾语，一般为概念，而 predicate 为谓词，一般为对象属性。如家族本体中，已知 A hasSibling B，B has Daughter C，则蕴含有 A hasNiece C，但基于前面几种内置的推理机无法实现这一功能，此处定义推理规则 rule_hasNiece，规则定义如下：

```
[rule_hasNiece: (?A family: hasSibling?B), (?B family: hasDaughter?C) →(?A family: hasNiece?C)]
```

通过以下语句即可以创建通用规则引擎：

```
Reasoner ruleReasoner= new GenericRuleReasoner(Rule.parseRules(rules))
```

8.6.2　在 Jena 中集成外部推理机

外部推理机可以通过两种方式集成到 Jena 中：在 Java 的 classpath 中直接添加.jar 文件或类文件，或者通过远程 DIG 接口来实现。一般通过设置 classpath 来实现外部推理机集成的方法比较方便高效，因为不需要任何网络来和推理机进行交互。Pellet 是基于 Java 的 OWL-DL 推理机，因此提

表 8-3 对样例本体进行 RDFS 推理后得到的蕴含信息

个体名称	推理得到的蕴含信息	说　明
Chihuahua	type:http://example.org#Breed	个体 Chihuahua 属于 SmallBreed 类，而 SmallBreed 类是 Breed 类的子类
GoldenRetriever	type:http://example.org#Breed	个体 GoldenRetriever 属于 LargeBreed 类，而 LargeBreed 类是 Breed 类的子类
Daisy	(1) name:Morning Daisy Bathered in Sunshine	(1) registeredName 属性是 name 属性的子属性
	(2) type:http://example.org#Mammal	(2) 个体 Daisy 属于 Canine 类，而 Canine 是 Mammal
Ryan	type:http://example.org#Mammal	个体 Ryan 属于 Human 类，而 Human 类似 Mammal 类的子类

表 8-4 对样例本体进行 OWL 推理后得到的蕴含信息

个体名称	推理得到的蕴含信息	说　明
Chihuahua	(1) type:http://www.w3.org/2002/07/owl#Thing (2) smaeAs::http://example.org# Chihuahua	(1) 任何类都是 owl: Thing 的子类 (2) 任何个体都与自身相同
GoldenRetriever	(1) type:http://www.w3.org/2002/07/owl#Thing (2) smaeAs::http://example.org# GoldenRetriever	(1) 任何类都是 owl: Thing 的子类 (2) 任何个体都与自身相同
Ryan	(1) type:http://www.w3.org/2002/07/owl#Thing (2) smaeAs::http://example.org# Ryan	(1) 任何类都是 owl: Thing 的子类 (2) 任何个体都与自身相同
Daisy	(1) hasOwner:http://example.org#Ryan (2) type:http://www.w3.org/2002/07/owl#Thing (3) smaeAs::http://example.org# PetOfRyan (4) smaeAs::http://example.org# Daisy	(1) ex:owns 属性和 ex:hasOwner 属性是逆反属性 (2) 任何类都是 owl: Thing 的子类 (3) Daisy 是 Mammal 类的个体，且 hasOwner 属性值为 Ryan (4) 任何个体都与自身相同
Amber	(1) type:http://www.w3.org/2002/07/owl#Thing (2) smaeAs::http://example.org# Canine	(1) 任何类都是 owl: Thing 的子类 (2) ex:Canine 类与 breed 属性取值为 Breed 对象的类等价

供了非常方便的包含.jar 的接口，非基于 Java 的推理机通过提供 JNI 接口也可以提供这样的访问。由于 Pellet 几乎支持 OWL1 和 OWL2 的所有特征，是一款合理完备的推理机，且对 SWRL 提供很好的支持，所以通过叠加 Pellet 可以明显提高 Jena 推理的完备性，弥补了单纯使用 Jena 推理机带来的不足。

对 Pellet 推理机的集成方式和 Jena 内部推理机很类似，使用 PelletReas0nerFact0ry. theInstance( ). create( ) 创建一个 Jena 推理机对象，推理机对象绑定到模式的方式也和 Jena 内部推理机一样。示例代码为：

```
Reasoner reasoner= PelletReasonerFaetory. thelnstance( ) . create( ) ;
reasoner= reasoner. bindSchema( ontModel) ;
InfModel infmodel= ModelFaetory. createInfModel( reasoned, ontModel) ;
```

如果一个推理机不允许通过 Java 的.jar 文件来直接访问，则 Jena 框架还支持 DIG 接口。这是一种标准的推理机接口，是关于通过 HTTP 协议访问描述逻辑处理的一个 XML 标准。叠加了 Pellet 推理机后，以上家族本体中的 hasNiece 规则，可以写成以下的 SWRL 规则：

```
hasSibling( ? , ? Y)^hasDaughter( ? Y, ? z)  → hasNiece( ?  , ? z)
```

实验和研究表明，通过推理机叠加，一方面利用了外部推理机 Pellet 对具有推理完备性和可判定性的 OWL DL 的支持，另一方面也充分发挥了自定义规则灵活广泛的优势[6~8]。这种叠加是目前使用开源软件能够实现的一种功能较强、结果较完备的语义推理机解决方案，但如果实现商业化的基于大型本体知识库的语义推理和应用，Racer 推理引擎则是更好的选择。

参考文献

[1]　纪兆辉. 本体的推理研究[J]. 南京师范大学学报. 2012，12(3)：54-59
[2]　潘超，古辉. 本体推理机及应用[J]. 计算机系统应用. 2012，19(9)：163-167

[3] 阎红灿,闫宏图,刘保相. Tableau 算法在粗逻辑知识推理中的应用. 贵州师范大学学报(自然科学版),2013.1(31):40-43
[4] 李海燕,李冠宇,韩国栓. 粗糙本体支持的知识推理框架[J]. 计算机工程应用. 2013,49(10):40-44
[5] 王松,李冠宇,李琳. 基于 SWRL 推理机制的研究[J]. 中国科技论文在线
[6] 阎红灿,王坚. 基于 P-集合的本体形式背景抽取. 计算机应用研究,2012,29(6):2196-2199
[7] 王坚. 粗逻辑在语义知识推理体系中的应用研究[D]. 河北联合大学,2011.12
[8] 汤怡洁,杨锐,李桂菊等. 基于知识资源的语义推理分析研究[R]. 2010.6

第 9 章　基于本体知识库的知识发现

目前在 Web 上进行的信息检索主要是利用搜索引擎，进行基于字符的关键字匹配检索。尽管搜索引擎在一定程度上避免了用户浏览网络信息的盲目性，给用户带来了便利，但是这种检索方式的主要问题是在返回大量不相关的结果的同时却又漏掉了一些相关页面，在很大程度上无法满足用户的需求。面对网络信息量的激增，传统的基于信息定位的 Web 信息表示方法，使得现有信息检索面临前文中所述的困难和窘境。因此，改进现有的信息检索技术，提高信息检索质量和效果的重要方法之一，就是以一种更容易被机器处理的表示方法来描述 Web 上的信息，而语义 Web 是目前一个较好的、相对与继续使用现有信息表述方法而开发文本处理技术更容易的、解决上述问题的选择，为语义检索和基于知识库的知识发现提供技术基础。因此，建立基于语义 Web 的知识发现方法具有实际意义和较高学术价值。

语义检索是将人工智能技术与自然语言处理技术结合，从语义理解的角度分别对信息资源、用户查询请求进行分析的信息检索方法，其检索方式是基于知识、语义的匹配。通过一系列词表控制法、概念空间、语义网络、语料库控制、语义本体等方法能够实现对概念与概念间语义的描述。词表控制法、概念空间、语义网络、语料库控制等方法都利用词汇在概念上的相关性，形成专业的手工词表或者机器学习生成的关联概念空间，这些方法一个明显的不足就是对概念关系的描述刻画程度有限。例如，主题词表只反映词间关系，无法表现公理、规则等，概念空间表达的多是概念间相关词的集合。这些方法在一定的程度上解决了基于关键字检索的不足，但只是一种初级的语义检索方法，不能更好地理解用户的检索意图和关键字所表达的语义。

语义本体比上述方法有更为丰富的语义表示途径，具有良好的概念层

次结构,通用的本体建模元语,利用概念之间的关系对概念语义进行表达,能很好地支持逻辑推理,在语义、知识层面描述信息资源的概念模型,适合知识的表达和网络环境下的知识共享和互操作,这些特性为语义检索提供了良好的框架结构和知识基础。基于领域本体的语义检索实质上是把借助领域本体规范后的检索请求按领域与标注后的信息源索引库进行语义匹配和语义推理,并提交给检索系统的过程[1]。基于领域本体对术语的严格描述和定义,以及反映术语间关系的语义网络,可以实现对查询请求和信息源知识的规范处理。把本体技术引入知识发现领域之后,就可以实现对某一领域内信息和知识的检索,较好地处理一义多词、一词多义所导致的问题,实现语义化的知识资源发现。

9.1　语义检索和知识发现

知识发现是一个从数据中发现知识的过程,其核心是数据挖掘,即通过聚类、关联规则分析等技术挖掘信息的频繁模式。基于知识库的知识发现核心是挖掘隐含的知识,通过语义检索和本体推理技术完成。

基于概念匹配的语义检索系统必须具备一定的知识体系来表达概念及概念间的逻辑语义关系,该知识体系以不同的类目分类(继承)而具有层次性,因不同的本体联想(语义关联)而形成一个语义网络。在知识层面或者在概念层面上建立的语义检索,能提供给用户一个缩小或扩大的检索范围,以获得某个概念的上位概念、下位概念以及平级概念等,通过概念的缩放,获得概念所对应的知识对象。另外采用文献[2]中所述的计算本体之间的相似度和相关度方法,在语义检索的同时计算本体之间的相似度和相关度,向用户提供有价值的参考信息,智能性地帮助用户进行有效的知识检索和知识导航。

本体形成了概念及其关系的分类化、层次化,使得语义检索能实现复杂的语义关系。本文按照从上而下的思想将知识的组织形式分成三层,概要分析如下:

（1）本体层：主要由分类本体构成了一个树形层次分类结构，形成分类概念空间，为基于概念匹配的语义检索提供基础；

（2）元数据描述层：主要对各种分布的知识对象进行元数据描述，建立知识对象的元素属性与值之间的对应关系，实现非结构、半结构化知识的结构化处理；

（3）网络资源层：是 Internet 和 Intranet 上具体的非结构、半结构化知识对象资源，也是知识检索最终要定位的具体资源。

三层结构的设计机理是：通过对类型层的概念空间进行概念匹配实现语义检索，由该概念在中间描述层发现与其链接的实例主题及元数据描述，由该实例主题通过 URI 找到具体的网络知识资源，从而实现从概念到具体知识对象的定位过程。

语义检索中最常遇见的难题是同义异形和同形异义关系的解决。同义异形关系的解决，可以采用如下两种方式实现：

方法一，在本体信息和知识对象标注时，在本体库中只对同义关系集合中的一个元素建立主题，通过在该主题上维护同义集合实现存储同义元素。这实质上是对同义集合中的元素赋予同样的标识，进行了统一化处理，从而屏蔽它们相互之间形式上的差异。在检索时，将同义集合中元素匹配检索项，以此形成同义检索的结果集；

方法二，在本体库中对同义关系集合中的元素建立同义关联。假如用户输入检索项，检索子系统首先对检索项进行概念匹配检索，其次是在同义关联中寻找同义项，然后对同义项再实现匹配检索，并最终形成同义检索结果返回给用户。

另外，对于后期形成的同义异形关系还可进行本体映射。采用不同的本体描述语言来组织知识的系统也将采用不同的方法来解决该问题，如基于 Topic Map 本体的知识检索系统使用第一种方法，而采用 RDF(S)、OWL 作为本体描述语言的系统通常采用第二种方法，如 OWL 中采用构造子 sameClassAs、samePropertyAs、equivalentTo 等形成了概念间的同义关系。

对于同形异义词，在知识组织时，可采用一定的限定词来界定词在特定

上下文中的含义，例如，在基于 TopicMap 本体的知识检索系统中，采用 Scope 来限定范围，如果某个主题的 Scope 是 computer software，则该主题代表计算机操作系统方面的文献；如果其 Scope 是 plane operation，则表示是操作飞机系统的文献资料。在基于 RDF(S)、OWL 等知识组织方式的本体检索系统往往通过概念的上下位关系来减少歧义干扰或者通过强制加上下文约束排除歧义的干扰。可见，知识检索的性能离不开有效的知识组织。

本体可以严格对概念进行描述和定义、反映概念之间的语义关系，是对某个领域知识的共同理解和描述，它为语义检索提供了良好的知识基础。在基于本体的语义检索模型中，对查询条件进行了语义层面的处理，表现为语义扩展。利用本体中定义的规范的概念作为索引项来对文档和用户查询进行描述，而利用语义索引项来表示文档和查询最重要的好处就是对用户的查询请求可以进行基于本体的语义扩展，从语义的角度对用户查询进行分析[3]。主要的本体扩展包括同义扩展、属性扩展、上位扩展、下位扩展、实例扩展等。

语义信息检索模型(见图 9-1)具有下述功能：

(1) 具有语义推理能力。一定的语义推理能力是基于本体语义检索系统相较于传统信息检索系统明显的进步，不但能够检索出明显的用户需求信息，而且还能根据一定的规则和公理推导出隐含的知识。

(2) 具有语义扩展能力。利用本体模型对检索式进行相关概念间关系的描述，对用户使用的查询概念，自动进行语义的扩展，即对查询概念的同义词、上位词、下位词进行相关的扩展、泛化等操作，这样可以提高查全率和查准率。

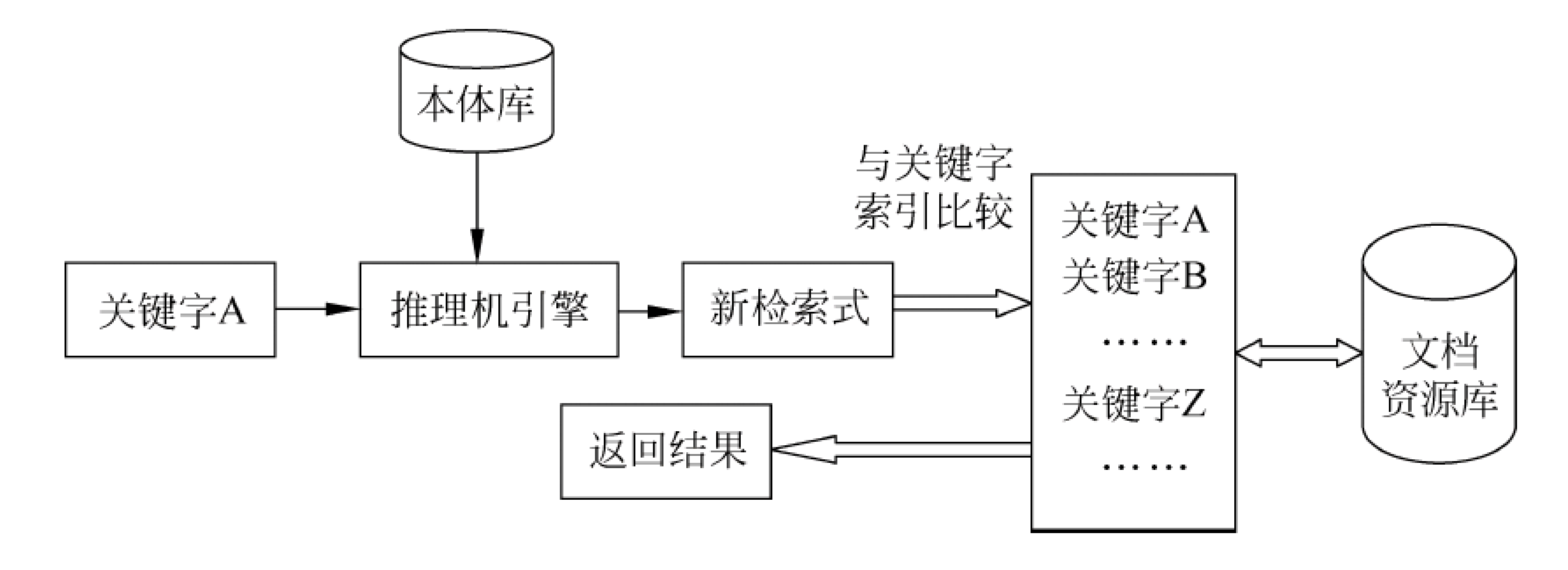


图 9-1 基于本体的语义检索模型

（3）传统的信息检索方法是将用户输入的检索关键字与信息资源库中的索引词按照字符进行匹配，检索目标结果的方法，检索系统只是把关键词作为一种标示符号、无法理解其所含有的语义信息。例如，用户检索“美洲住宿费标准”，传统的检索系统只返回含有“美洲住宿费标准”字样的检索结果，而只含“80 元/人・天”的检索结果将没有办法被系统直接检索出来。基于领域本体的语义检索系统的核心思想是加入领域本体作为进行检索匹配和推理的核心部件，与传统的检索方式相比，增加了本体推理层。本体层的作用过程如图 9-2 所示。

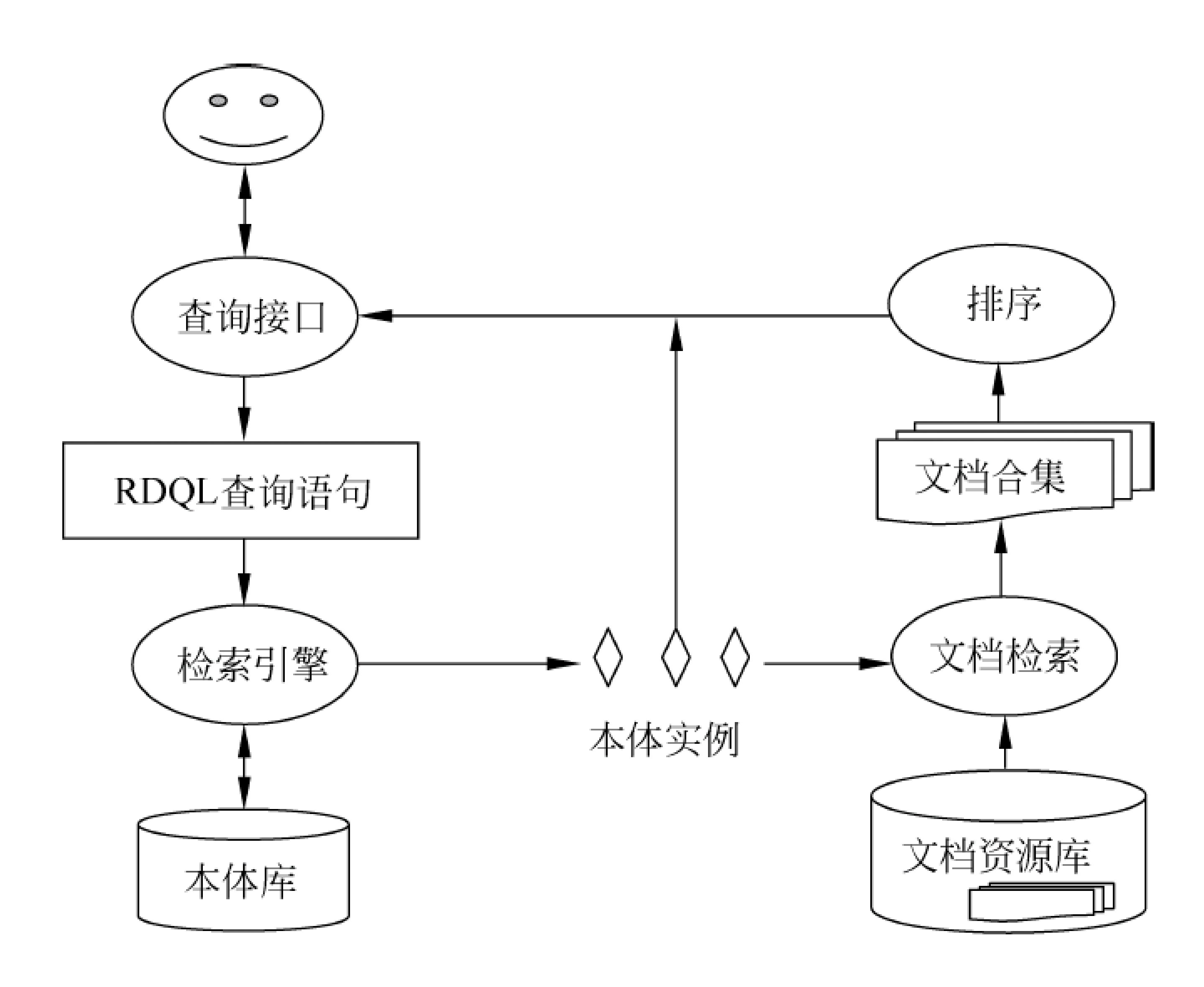


图 9-2　基于本体的语义检索-本体作用过程图

9.2　基于本体的语义检索关键技术

基于本体的语义检索的核心技术是语义推理，涉及推理机机制、推理规则生成和查询语言转换等关键技术。

9.2.1　基于描述逻辑的推理机

RACER 最早是由德国的 University of Hamburg 开发的，之后加拿大

的 Concordia University 和德国 University of Applied Sciences 也参与了开发。RACER 是一个知识表示系统，可以提供 DL 中 TBox 和 ABox 推理功能。同时 RACER 采用的算法是 Tableaux 演算，而 Tableaux 算法是一种有效的描述逻辑系统的推理方法。因此，使用基于 Tableaux 算法的 RACER 推理机，可以有效地处理基于描述逻辑的 OWL DL 本体。图 9-3 是 RACER 推理机的工作流程图。

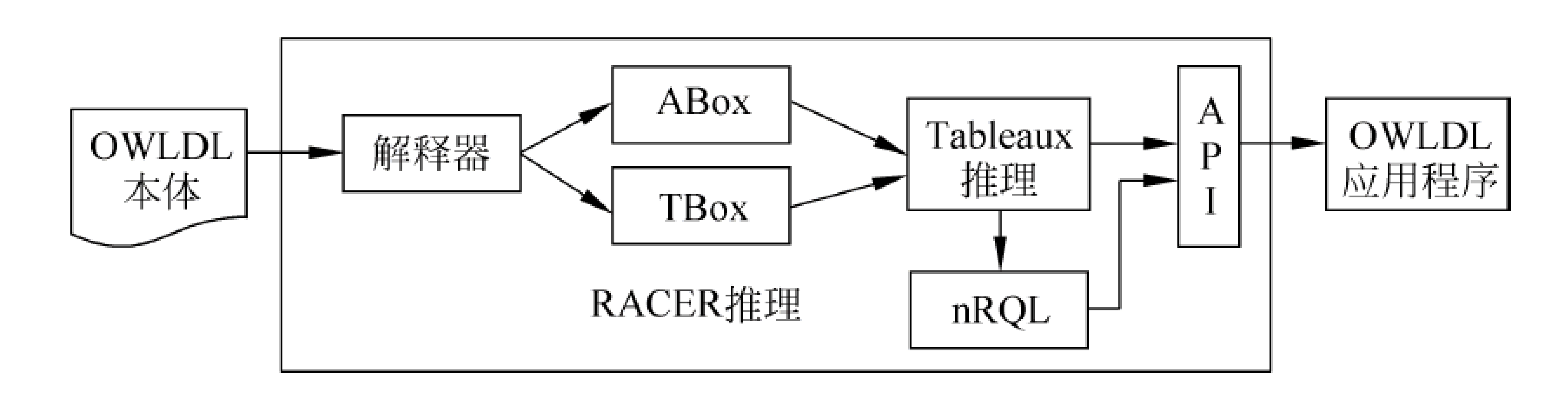


图 9-3　RACER 推理机的工作流程图

如图 9-3 所示，首先将 OWL DL 本体输入，经过 RACER 的解释器可以解析为 ABox 和 TBox。其中 ABox 主要由三元组组成，即主体、谓词和客体，同时任何一个主体或者客体又可以是其他三元组的主体或者客体。将解析后的 ABox 和 TBox 进行 Tableaux 推理。RACER 推理系统可以使用 JRacer 的 Java API 来与 RACER 系统进行 TCP 通信，同时可以使用 nRQL (new Racer Query Language)获取实例的解析数据。JRacer 中的 Java API 可以方便实现 RACER 和 nRQL 到 Java 类的转化。OWL 本体通过 RACER 的内部解析机进行本体的解析后，通过相应的 API 实现与其他程序的接口。

描述逻辑推理机 RACER 的推理功能如下：

(1) TBox 的一致性检测；

(2) TBox 的包含关系检测；

(3) 发现 TBox 中定义的所有概念是否存在不一致；

(4) 确定 TBox 的一个概念的双亲和孩子关系；

(5) 检测一个 TBox 与 Abox 是否一致。

9.2.2　基于规则的推理机

Jena 是由美国惠普实验室（HP Laboratory）为语义网研究项目开发的一套开放源代码的数据包，目的是建立基于语义网的系统结构，使其可以在 Java 系统中解析数据信息，它为 RDF、RDFS 和 OWL 提供了一个可编程实现的环境，具有很好的稳定性和通用性。通过 Jena 提供的 OWL API 接口、SPARQL 查询接口和本体推理机接口，可以编写基于本体的数据集成、语义检索等智能应用程序。

Jena 目前包括两个版本：

（1）Jena1：它包含了对 RDF 的操作的 API 和 RDQL；

（2）Jena2：能够处理 OWL、DAML＋OIL 的 API，是 Jena 的最新版本。这里提到的 Jena 都指的是 Jena2。Jena 框架中的推理机主要由如图 9-4 所示的部分构成，这些组成部分是处理语义 Web 环境下的语义检索的重要组成部分。

Jena 的各个组成部分包括：

（1）ARP（Another RDF Parser）：ARP 当前的版本是 ARP2，这个 RDF 解析器在最新的 RDF 标准上进行了改进，ARP 在 Jean 中的作用是解析 RDF/XML 格式信息文件，即 Jena 中的 RDF 文件的获取，当然也能够独立于 Jena 而应用到其他 Java 程序中，但是要求 JRE1.4 以上的版本；

（2）RDF API：RDF API 主要功能是对 RDF 模型进行相应的处理，在 Jean 中有很多接口来访问 RDF 模型，主要是 com.hp.hpl.jena.rdf.model 包；

（3）Ontology API：Ontology API 主要用来为 OWL、DAML＋OIL 和 RDFS 等以 RDF 格式存储的程序开放提供支持，主要是 com.hp.hpl.jena.ontology 包；

（4）推理子系统：主要功能是通过引入相关推理引擎完成对 RDFS 和 OWL 的规则推理，用户也可以自定义一些规则。推理子系统是与 Ontology API 相互紧密连接的，使其能够通过推理得到一个特定的本体源的更多信息，主要是 com.hp.hpl.jena.reasoner 包。

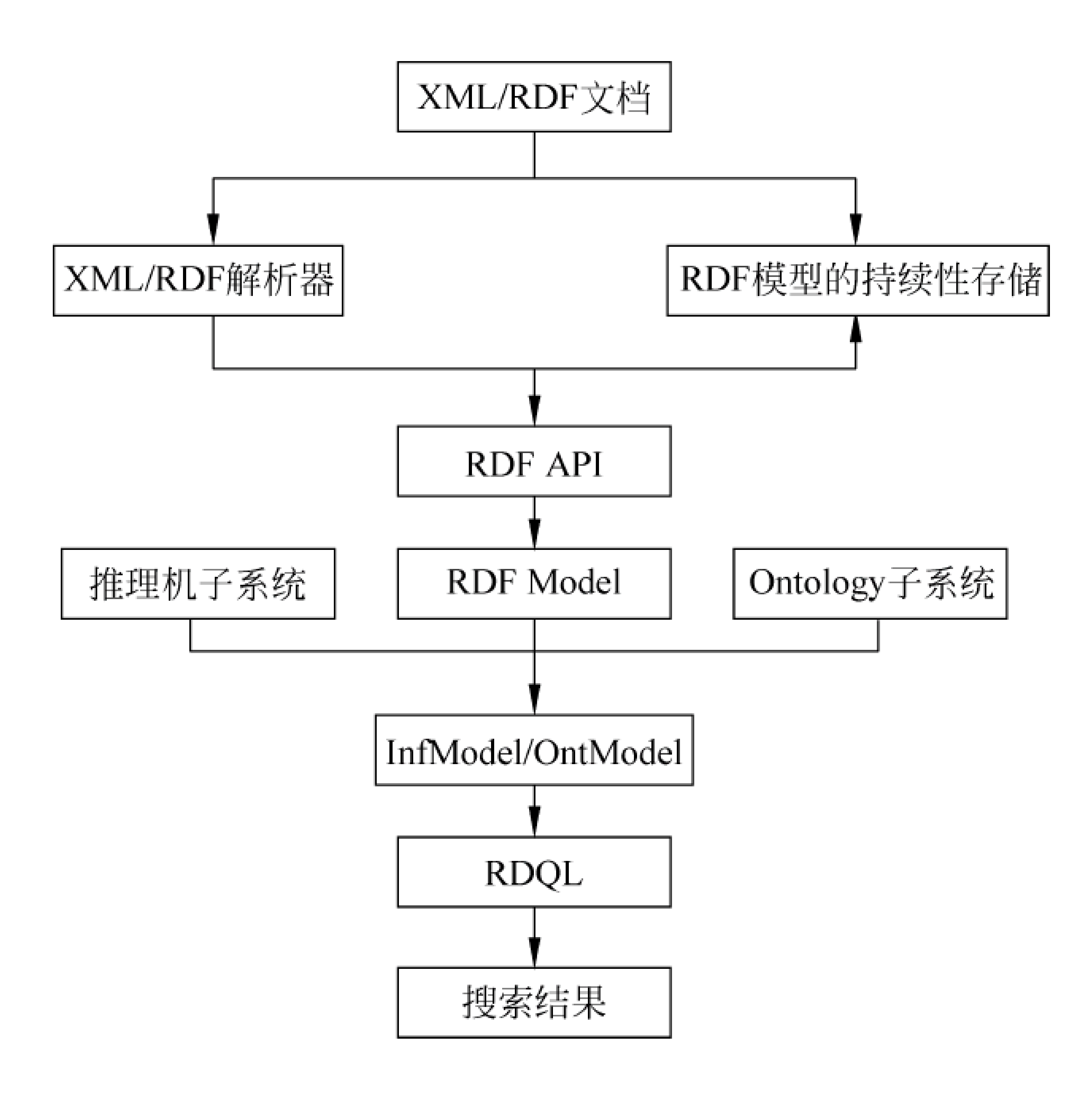


图 9-4　Jena 的组成部分

若需要在 Jena 中获取一个推理机，可以选择以下几种方法：

（1）工厂方法：每个推理机有含有推理机工厂（Reason Factory）类，一个实例 Reasoner 是通过调用推理机工厂类的静态方法 getInstance( ) 获得。

（2）推理机注册器：Jena 提供了一个全局的推理机注册（Reasoner Registry），推理注册机根据已经创建或读入 RDF 三元组描述的信息资源和 Ontology 内包含的信息利用规则创建出推理机。

（3）Jena 自带的基于一般规则的推理机：这种形式的推理机是在自定义规则的基础上实现的。推理机对推理的完成是通过一些可以解释推理规则的引擎。具体的引擎有前向链引擎、后项链引擎以及混合式规则引擎。至于推理规则，用户可以按照自身具体要求来制定个性化规则，再根据这些规则来选择特定的推理机，使用第三方推理机。当然，也可以使用 Jena 之外的推理机，他们可以和 Jena 集成。

（4）本体共性规范（OntModelSpaec）方法：当用户用 Ontology API 时，

用户能够指定一个模型规范(OntModelSpaec)赋予一个模型,即同时能够建立一个与之对应的推理机。

9.2.3　推理规则语言 SWRL

本体用于语义查询系统的一个重要原因在于本体是建立在逻辑基础上的,这使得数据源中的一些隐含的概念或关系可以被发现,也就决定了基于描述逻辑的本体能够支持对用户查询的推理处理。但是本体的语义表达能力还是有局限性的,其表达能力局限于描述逻辑,不能表示一般形式的规则,更不能表示涉及时空关系的连续变化的事件流或是基于统计数据的不确定知识。

2003 年 11 月 DARPA 组织提出了一套语义 Web 规则语言 SWRL。SWRL 是集本体和规则于一体的一种语言,由 RuleML 演变而来,一样是 XML Based 的规则格式,具有人机可读的优点,可较为清楚地表现规则之间的关系。此外 SWRL 可以引用本体中的元素来编辑规则,这是和 RuleML 的不同之处。SWRL 在 OWL 中加入了规则,因为规则能够提供更强的逻辑表达能力,OWL 则不能达到这样的能力。尽管 SWRL 刚刚被研究,但是一阶逻辑已经被很充分地研究过了。另外,结合一阶逻辑也使得 SWRL 可以容易地与传统的关系数据库进行交流。

本体的构造采用 OWL 语言,它是语义检索的前提,参数约束及规则采用 SWRL,其表达要依靠 OWL 语言来建立基本的术语及关系,所以规则的建立采用的是 OWL+SWRL 的形式[4-5]。

对本体中大部分的内容参数约束可以使用 OWL 中的值约束、基数约束等表达,但是外部参数、隐含规则,或选择表达式,例如属性的组合、限制假设,仅用 OWL 语言难以表达,需要借助 SWRL。SWRL 主要由 Imp、Atom、Variable 和 Building 组成,其框架结构如图 9-5 所示。

Imp 是 SWRL 的规则部分,包括 head 和 body,其中 head 表示推理的结果,body 表示推理前的状态。head 和 body 所使用的 instance 是由 Atom 或 Variable 提供的。

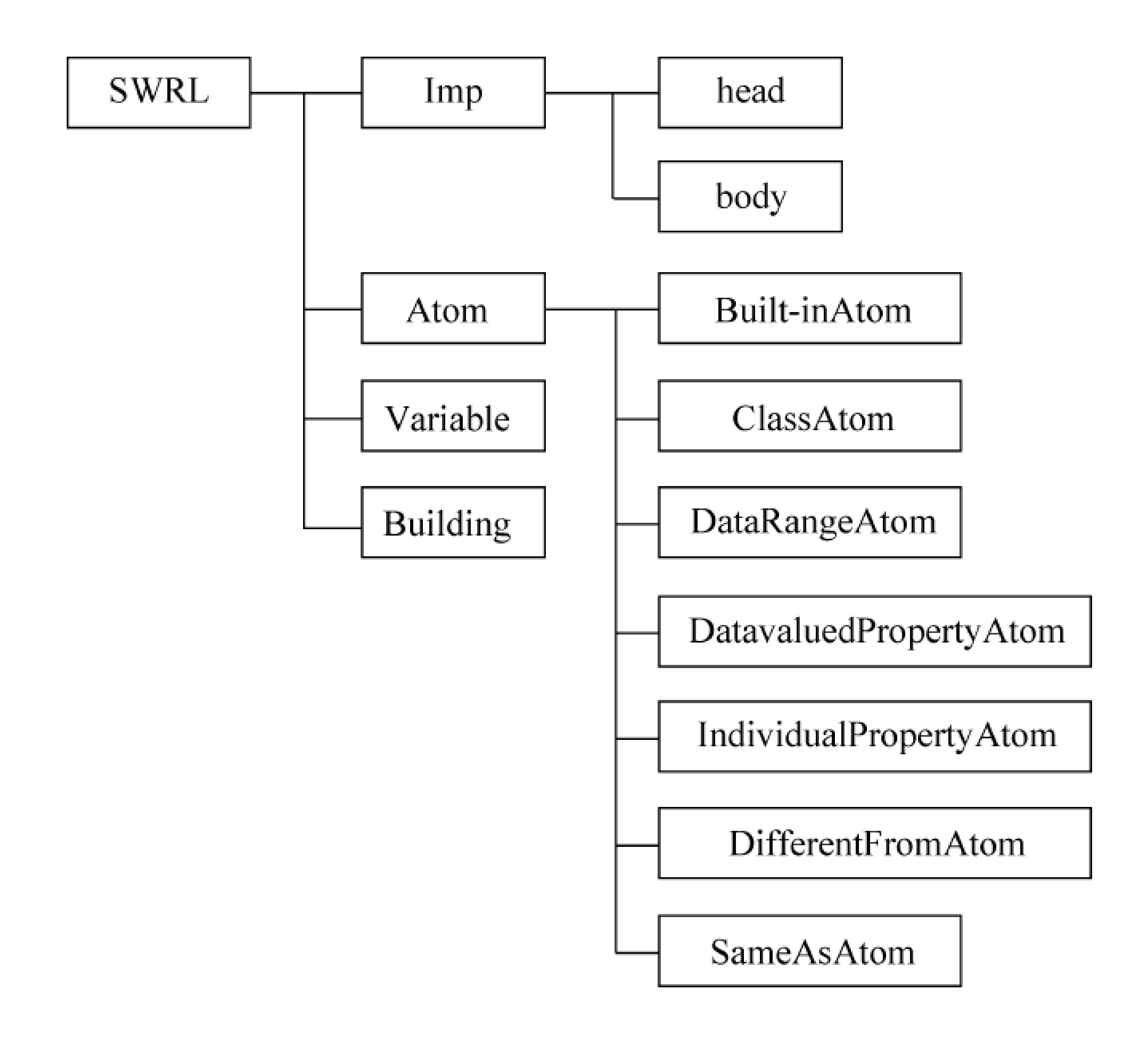


图 9-5　SWRL 语言架构

Atom 是 head 和 body 允许出现的基本成分，在推理的结果 head 中只允许出现一个 Atom，而在推理前的状态 body 中，可以是多个 Atom 的合取。在 SWRL 中，利用本体中的实例和属性来建立 Atom 子句。其中本体的实例作为 Atom 的参数，本体的属性作为 Atom 的属性。图 9-6 说明了 Atom 与本体的关系。

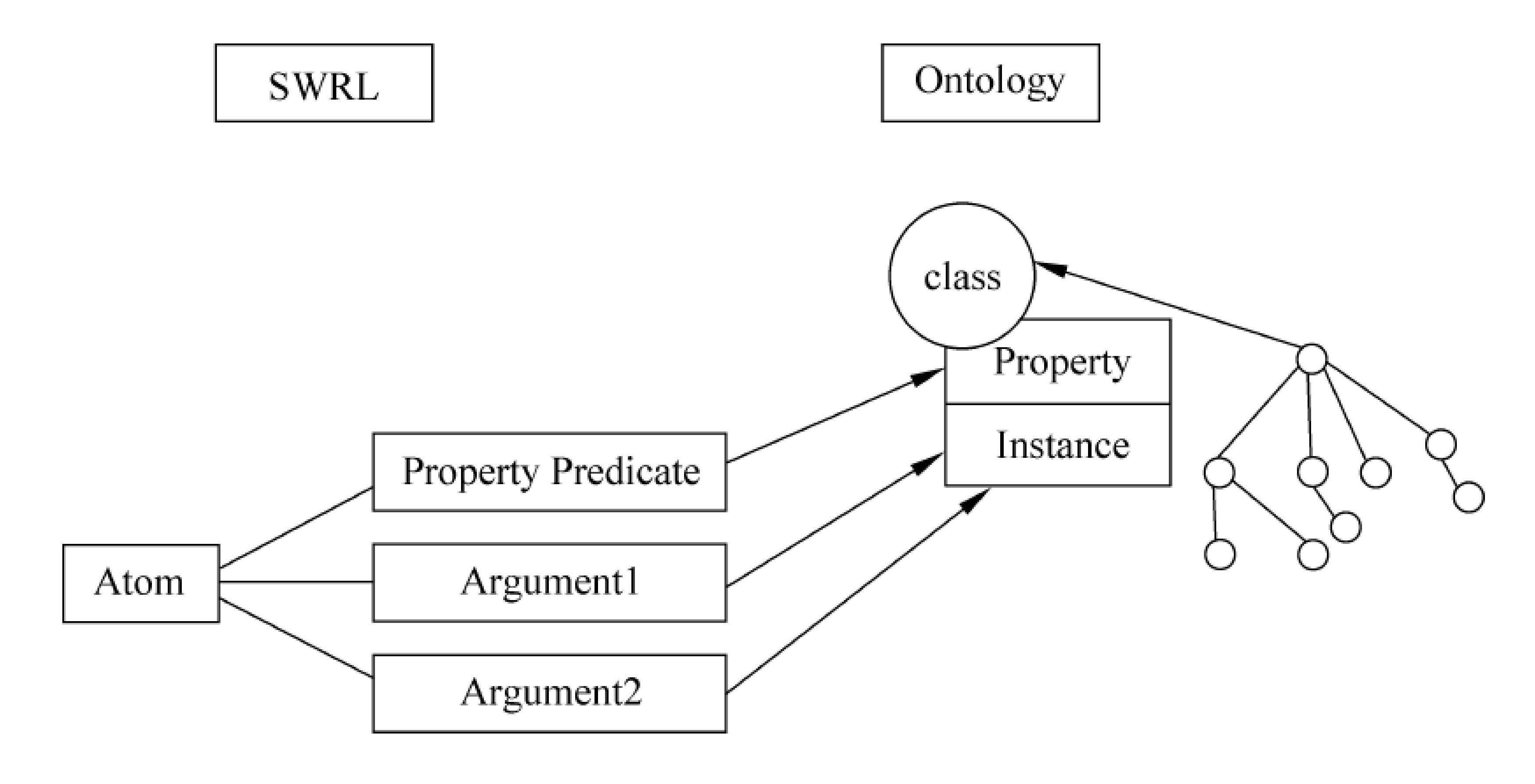


图 9-6　Atom 和本体关系

Variable 用于定义 Atom 中使用的变量，在 Atom 包含许多限制式，主要有以下四种：

（1）C（x）：C 是 OWL 描述；

（2）P（x，y）：P 是 OWL 的属性，x，y 可以是变量、OWL individuals 或 OWL data valuea；

（3）SameAs(x，y)：x 和 y 相等；

（4）DifferentFrom(x，y)：x 和 y 不同。

Building 用于定义 SWRL 中的各种逻辑比较关系，它们分别用于数值比较、数学计算、布尔运算、字符串操作、时间和日期、URIS 和列表，等等。其中用于数值比较的 Building 参见表 9-1。

表 9-1　SWRL Building 数值比较

swrl:Building	Example
Equal	Argument＝Argument2
notEqual	Argument≠Argument2
lessThan	Argument＜ Argument2
lessThanOrEqual	Argument≤Argument2
greaterThan	Argument＞ Argument2
greaterThanOrEqual	Argument≥Argument2

SWRL 的表示方式主要有两种，如图 9-7 所示。一种是 XML 的表达方式，以 RuleML＋OWLX 的方式描述；另一种是 RDF 的表达方式，以 OWL＋RDF 的方式描述。

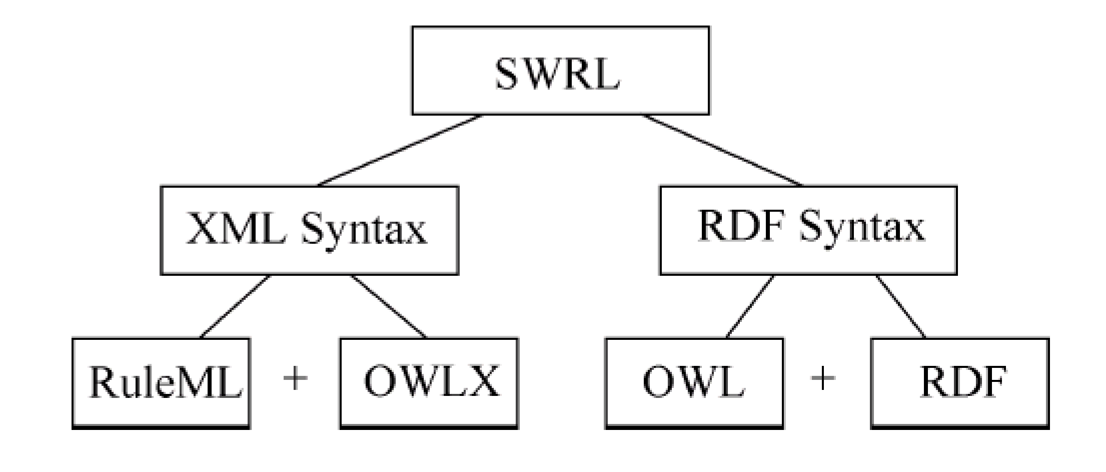


图 9-7　SWRL 的表示方式

以 XML 作为表达方式的优点：

（1）任意 OWL 的类可以作为谓词(predicates)在规则中出现；

（2）规则和本体的公理(axiom)可以混用；

（3）可以通过 XSLT 将 OWLX 转换为其他语法；

（4）现存的 RuIeML 工具可以继续编辑 SWRL。

RDF 方式则是直接使用语义网的标准语言进行描述，因为能直接针对 OWL 所建立的本体结合，不需要经过其他的转换，SWRL 中的格式也由 OWL 格式所规范。以这种方式所设计的规则，最大的好处在于内部的变量都是 RDF 的方式表示，变量本身就是 RDF 的形式，可使得将变量对应到本体的工作较为简单。而以 RuIeML 的方式表示，变量只是单纯的文字比对，这些变量本身并不带有实际资源的意义。

SWRL 的两种表示方式的语法结构是完全相同的，只是表达方式的不同。规则是以 SWRL 语法格式进行描述的，SWRL 规则本身建立在 OWL 本体之上，可以结合已有的 OWL 知识库中的信息来建立 SWRL 规则库，它和 OWL 知识库是整个推理框架的核心。

9.2.4 语义查询语言的转换

在语义检索中，检索者必须熟悉本体的语法、形式化查询语言、目标本体的结构和词汇等，才能完成检索。为了方便检索，输入采用自然语言。因此，在这个阶段需要将自然语言转换为语义查询语言 SPARQL[6-7]。

SPARQL (Simple Protocol and RDF Query Language)，是为 RDF 开发的一种查询语言和数据获取协议，是为 W3C 所开发的 RDF 数据模型所定义，但是可以用于任何可以用 RDF 来表示的信息资源。SPARQL 协议和 RDF 查询语言(SPARQL)于 2008 年 1 月 15 日正式成为一项 W3C 推荐标准，它将 Web2.0 和 Semantic web 两种新的 web 技术联系起来，很有可能成为将来的主流网络数据库的查询语言和数据获取标准。

SPARQL 查询语言是在 RDF 查询语言(如 rdfDB，RDQL 和 SeRQL)基础之上构建的功能更强大的 RDF 查询语言。它包括两个独立的部分：查询语言规范、SPARQL 语言的数据访问协议和返回查询结果的 XML 格式。SPARQL 查询语言可以完成下述任务：

（1）通过 URIs 形式、无格式字符、空白结点等形式提取相关信息；

（2）可以提取 RDF 子图；

（3）在 SPARQL 查询图的基础上构建新的 RDF 图。

SPARQL 查询语言为语义网上的用户提供了查询语言类似于关系数据库用户所使用的查询语言。研究中认为一个 RDF 图是可以看成是一个二段式结构，包括主语、谓语和宾语。SPARQL 查询语言的实质是基于图模式匹配的查询语言即二段式结构模式，与 RDF 二段结构很相似，但是 SPARQL 查询的图模式在主语、谓语或宾语部分用变量代替了 RDF 术语。将这些二段结构组合就能形成一个基本图模式，为了形成 SPARQL 可以识别和操作的模式，这些图必须进行严格的匹配。SPARQL 查询语言的核心语句包括两个部分：负责识别查询结果中的变量的 Selec 子句，和子句有一个二段式结构模式的 where 子句。

在 Jena 中使用 SPARQL，可以通过叫作 ARQ 的模块实现。除了实现 SPARQL 之外，ARQ 的查询引擎还可以解析使用 RDQL 或者它自己内部的查询语言表示的查询。ARQ 的开发很活跃，可以从 Jena 的 CVS 仓库或者自包含的下载文件中获得它。下载最新的 ARQ 发行包，对其进行解压，把环境变量 ARQROOT 设置成指向 ARQ 目录即可。还需要恢复 ARQ bin 目录的读取和执行权限。如果把 bin 目录添加到执行路径中非常方便，那是因为它包含从命令行调用 ARQ 的包装器脚本。

Java 应用程序可以直接调用 Jena 的 SPARQL 功能。通过 com. hp. hpl. jena. query 包中的类，使用 Jena 来创建和执行 SPARQL 查询。使用 QueryFactory 是最简单的方法。QueryFactory 有各种 create()方法，用来从文件或者 String 读取文本查询。这些 create()方法返回 Query 对象，这个对象封装了解析后的查询。

下一步是创建 QueryExecution 的实例，这个类表示查询的一个执行。要获得 QueryExecution，通过调用 QueryExecutionFactory. create(query, model)，并传入要执行的 Query 以及查询要处理的 Model。QueryExecution 有几种不同的执行方式，可以调用 exeSelect()，该方法将返回一个

ResultSet。ResultSet 在查询返回的每个 QuerySolution 上进行迭代，提供了对每个绑定变量值的访问。此外，还可以使用 ResultSetFormatter，以不同格式输出查询的结果。因为查询的数据是编程方式提供的，所以查询不需要 FROM 子句。

SPARQL 允许以 XML 格式返回查询结果，采用的格式叫作 SPARQL 变量绑定结果 XML 格式。这个用 Schema 定义的格式是 RDF 查询和 XML 工具及库之间的桥梁。这项功能还有许多潜在用途，可以把 SPARQL 查询的结果通过 XSLT 转换成 Web 面或 RSS feed，通过 XPath 访问结果，或者把结果文件返回给 SOAP 或 AJAX 客户。要以 XML 格式返回查询结果，请使用 ResultSetFormatter. outputAsXML()方法，或者在命令行指定 results rs/xml。

9.2.5 语义相似性排序

文献[6]提出的基于特征的语义匹配规则，即：

$$
\begin{aligned}
&\mathrm{Sim}(\mathrm{A},\mathrm{B},\mathrm{O}) \\
&= u\frac{n(C_{super}(A,O)\cap C_{super}(B,O))}{n(C_{super}(A,O)\cap C_{super}(B,O))+\min\{P_{super}(A,B,O)\}} \\
&\quad + v\frac{n(C_{sub}(A,O)\cap C_{sub}(B,O))}{n(C_{sub}(A,O)\cap C_{sub}(B,O))+\min\{P_{sub}(A,B,O)\}}+\varphi\frac{P_{sat}}{P_{tot}}
\end{aligned}
$$

其中 $C_{super}(A,O)$ 和 $C_{super}(B,O)$ 表示在本体树 O 中，包含 A 和 B 的概念数量；分子 $n(C_{super}(A,O)\cap C_{super}(B,O))$ 表示交集的个数，分母中 $\min\{P_{super}(A,B,O)\}$ 表示 A、B 的概念在本体图中的最短路径。公式中的 u、v、φ 为三个项的系数，其中 u 表示概念 A、B 相对于本体树 O 父概念的相似度的系数，v 是概念 A、B 相对于本体树 O 子概念的相似度的系数，φ 是参数约束满足的系数，如果参数很多，φ 的值就相对大一些，三个参数的和为 1，即 $u+v+\varphi=1$。

在上述公式中 $0<Sim(A,B,O)<1$，当 $A\equiv B$ 时，$Sim(A,B,O)=1$，当完全不相关时，或者没有共有的父概念和子概念以及共同的参数时，$Sim(A,B,O)=0$。

9.3　基于知识发现的案例推理应用模型

人们为了解决一个问题，常常先从记忆的知识库中寻找所有与新问题相关的经验和知识，然后把其中相关程度高的部分整合运用到新问题的求解过程中，这就是基于案例推理(case-based reasoning，CBR)过程的求解原理。随着人们对 CBR 研究和应用的逐渐深入，使得 CBR 的应用范围和领域不断扩大。先后在通用问题求解、法律案例、医药医疗、天气预报、机器故障诊断及企业咨询决策等方面获得应用，在网络和电子商务方面的应用也在不断展开。

案例推理的核心是案例匹配，对案例的检索最终将转化为案例中属性概念集的检索。为了实现案例的语义检索和检索结果案例的相关性排序，首先要解决案例的特征属性之间的关系表示和存储的问题。本文提出的案例推理模型中采用本体技术构建知识库，并通过本体推理实现案例检索的语义推理。

案例检索的质量直接关系到案例推理的质量，案例检索中最重要的是案例间的相似度计算，我们将集对分析理论引入案例的相似度计算中，建立了基于集对分析的案例检索模型[8-9]。

9.3.1　案例相似度计算

集对分析的重要工具是联系数，而联系数的确定与集对间的同一性、差异性及相反性有关，比对集对分析，定义了与案例有关的集对联系度，案例集对及其联系数。

在进行案例检索时，对案例间的相似性度量可以归结到对案例属性的相似性度量，案例库中往往存有大量的案例，案例属性类型更是种类繁多，因为人为因素，有些案例属性的数据可能被遗漏，不同的属性类型不同人的操作可能造成属性的不确定性、模糊性，而集对分析可以有效地处理不确定性系统，本部分利用集对分来处理案例系统间不确定性。

为了将集对分析引入案例推理系统中，结合案例检索的特点和集对分析的主要思想，给出以下定义：

定义 9-1 集对联系度：在问题 W 的背景下探讨集对 H 的性质进行，假设提取出 N 个特性，其中在 S 个性质上具有同一性，在 P 个特性上相反，在其余的 $F=N\text{-}S\text{-}P$ 个特性上既不对立，又不同一，即其性质不确定，则称比值：

S/N 为集对同一度；

F/N 为集对差异度；

P/N 为集对相反度，

式子 $u(w)=\frac{S}{N}+\frac{F}{N}i+\frac{P}{N}j$ 称为集对 H 的联系度，

i 为差异度标记符号，根据不同的情况在[−1,1]之间取值；j 是对立度标记符号，规定 $j\equiv-1$，为表示方便，式子可简写为：

$$u = a + bi + cj \tag{9-1}$$

由以上定义可以看出，a，b，c 三个数满足 $a+b+c=1$。

定义 9-2 案例集对：问题案例 q 和案例库中的每一个案例 p 之间存在一定的映射关系，这种映射关系将其构成案例集对。并且用式子(q,p)来表示问题案例 q 与案例库中某一案例 p 构成的案例集对。

定义 9-3 案例集对的属性集对：问题案例 q 与某一案例 p 关于同一属性的属性值构成案例集对的属性集对。

例如，假设案例集对(q,p)与 n 个属性有关，分别为：x_1，x_2，…，x_n，案例 q 与案例 p 关于这 n 个属性的属性值分别为 x_{q1}，x_{q2}，…，x_{qn} 和 x_{p1}，x_{p2}，…，x_{pn}，则(x_{qi}，x_{p1})，(x_{q2}，x_{p2})…(x_{qn}，x_{pn})均为案例集对(q,p)的属性集对。

定义 9-4 属性的联系数：案例集对(q,p)的属性集对间的联系度表达式称为属性的联系数。

例如，用表达式 $u_l=a_l+b_li_l+c_lj_l$ 表示案例集对(q,p)关于第 l 个属性的联系数，因为属性值只有一个，所以表达式只有一项，例如两个案例在第 l 个属性上具有同一性，那么联系数记为：$u_l=a_l$ 且 $a_l=1$，若具有差异性则记为：$u_l=b_li_l$ 且 $b_l=1$。

这样，在计算案例属性联系数时，对于有属性值缺失的情况，根据集对分析理论，认为两者具有差异性，即性质不确定，有效处理了推理中不确定性信息问题。

在本文实证研究的中医喘证案例中，从案例库中获取案例的属性特征，假设与 n 个属性有关，分别为：$x_1, x_2, \cdots, x_n$，然后把问题案例 q 与案例库中每一个案例 p 的 n 个属性分别分析比较，确定案例集对(q，p)的每一个属性集对的联系数：

$$u_l = a_l + b_l i_l + c_l j_l, \quad i_l \in [-1,1], \quad j_l = -1, l = 1,2,\cdots,n \quad (9\text{-}2)$$

其中，a_l 表示问题案例 q 与某案例 p 关于第 l 个属性的同一度；b_l 表示它们之间的差异度；c_l 表示它们之间的相反度，且 $a_l + b_l + c_l = 1$，那么问题案例 q 与案例库中某一案例 p 间关于 n 个属性集对的联系数分别为：u_1，$u_2, \cdots, u_n$，这是计算两个案例间相似程度时非常重要的一部分。

假设在案例集对(q，p)，$x_1, x_2, \cdots, x_n$ 是选取的重要概念的属性，它们的重要度分别为 $w_1, w_2, \cdots, w_n$，则结合(9-1)和(9-2)式，得到问题案例 q 与某案例 p 之间的案例相似度，记为：

$$\begin{aligned} \mathrm{Sim}(p,q) &= \sum_{l=1}^{n} a_l w_l + \sum_{l=1}^{n} b_l w_l i + \sum_{l=1}^{n} c_l w_l j \\ &= \mathrm{A} + \mathrm{B}i + \mathrm{C}j \\ i \in [-1,1]\ j &= -1, \quad l = 1,2\cdots,n \end{aligned} \quad (9\text{-}3)$$

其中 $A = \sum_{l=1}^{n} a_l w_l, B = \sum_{l=1}^{n} b_l w_l i, C = \sum_{l=1}^{n} c_l w_l$

给定合适的 i 值，可以得出问题案例 q 与某案例 p 的相似度值，按照相似度值顺序存入目标案例库，计算结果大于设定阈值的案例即为我们要找的目标案例。在有些案例中，特别是医学案例，由于古医案描述的模糊性，人为的理解差异大大影响了数据的确定性，因此需要考虑相似度的主观性认识，一般原则是：选取的案例应满足，在两案例的联系度式子中，同一度 A 尽可能大，对立度 C 和差异度 B 尽可能的小。

除了案例相似度计算方法，案例的表示方法、存储方式以及属性重要度的选取对案例的检索质量都有影响。

通常计算联系度时通过加权平均的方法尽量减少权系数的影响，但仍不能去除人为定制权系数的弊端，为了减少人为偏好，本文将案例属性在本体知识库中的层次作为体现属性重要度的一个参考，层次高的属性重要度比层次低的属性重要度要高。

9.3.2 案例匹配的语义检索

利用本体概念间的关系进行语义推理，实现对关键字的语义扩展查询检索。查询扩展主要是针对关键词的语义信息进行扩展，主要包括关键词语义间的包含、传递等关系，图 9-8 展示了通过本体推理实现语义检索的案例匹配过程。推理规则选择 SPARQL 本体查询语言，根据中医喘证学科的概念特征对本体推理机定义了如下的规则：

(1) Rule1：(?x has symptom ?y)，(?x subclass of ?z)→(?z has symptom ?y)

如果某类型喘 x 具有症状 y，某类型喘 x 是某类型喘 z 的子类，那么某

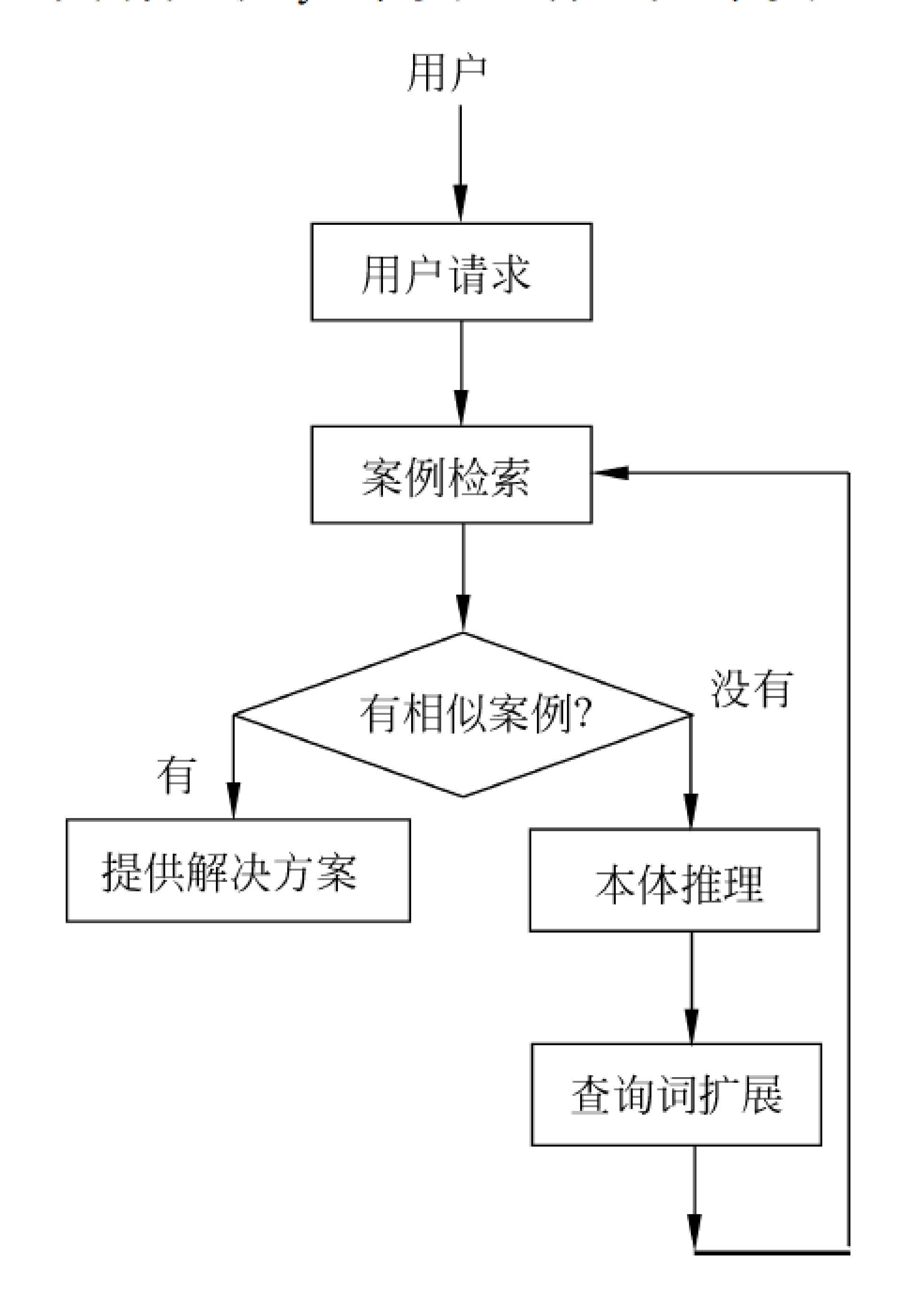


图 9-8 基于语义检索的案例匹配过程

类型喘 z 具有症状 y；

(2) Rule2(?x cause ?y), (?x subattribute of ?z)→(?x cause ?y)

如果 x 导致某类型喘 y，x 是某类型喘 z 的一个属性，那么 x 可导致 z；

(3) Rule3(?x cause ?y), (?z subclass of ?x)→(?z cause ?y)

如果因素 x 可以导致某类型喘 y 的发生，因素 z 是因素 x 的子类，那么因素 z 也可导致某类型喘 y 的发生。

推理核心代码：

```
OntModel model =ModelFactory. createOntologyModel ( Ont-ModelSpec. OWL_
MEM, null ) ;
    model. read( new FileInputStream( "D:\\ontology\\information. owl") , "") ;
    Model base = maker. createModel ( " model");
    //创建一个默认模型 model
    OntModelSpecspec = new OntModelSpec( OntModelSpec. OWL_MEM) ;
    OntModel newmodel = ModelFactory. createOntologyModel( spec, base) ;
    //定义推理规则
    String caus rule =" Rule1:( ?x has symptom ?y),(?x subclass of ?z)→(?z has
symptom ?y)" +" Rule2(?x cause ?y), (?x subattribute of ?z)→(?x cause ?y)" "
+" Rule3(?x cause ?y), (?z subclass of ?x)→(?z cause ?y);
    //查询语句
    String queryString ="PREFIX Info: < http://www. owl-onto-logies. com/
Information. ow l# > " +" SELECT ? case ? subject " + "WHERE { ? case
Info:cause?
subject } " ;
    //根据自定义规则创建推理机
    Reasoner cause_reasoner = new GenericRu-leReasoner(Rule. parseRules( caus_
rule) ) ;
    //绑定实例文件与推理机
    InfModel inf = ModelFactory. createInf-Model( caus_reasoner, newmodel) ;
    Query query = QueryFactory . create( queryString) ;
    //执行查询
    QueryExecution qe = QueryExecutionFactory.create( query, inf) ;
    ResultSet results = qe. execSelect ( ) ;
```

9.3.3　案例推理过程

图 9-9 展示了案例推理中检索词扩展、构造属性集对、属性联系度计算、案例相似度计算的处理过程，其所对应的检索步骤为：系统根据用户输入问

题案例的描述信息，通过本体知识库进行近义或同义词的扩展，抽取出与问题案例有关的属性集合，从而构造案例集对的属性集对，应用公式(9-3)计算案例间的相似程度。

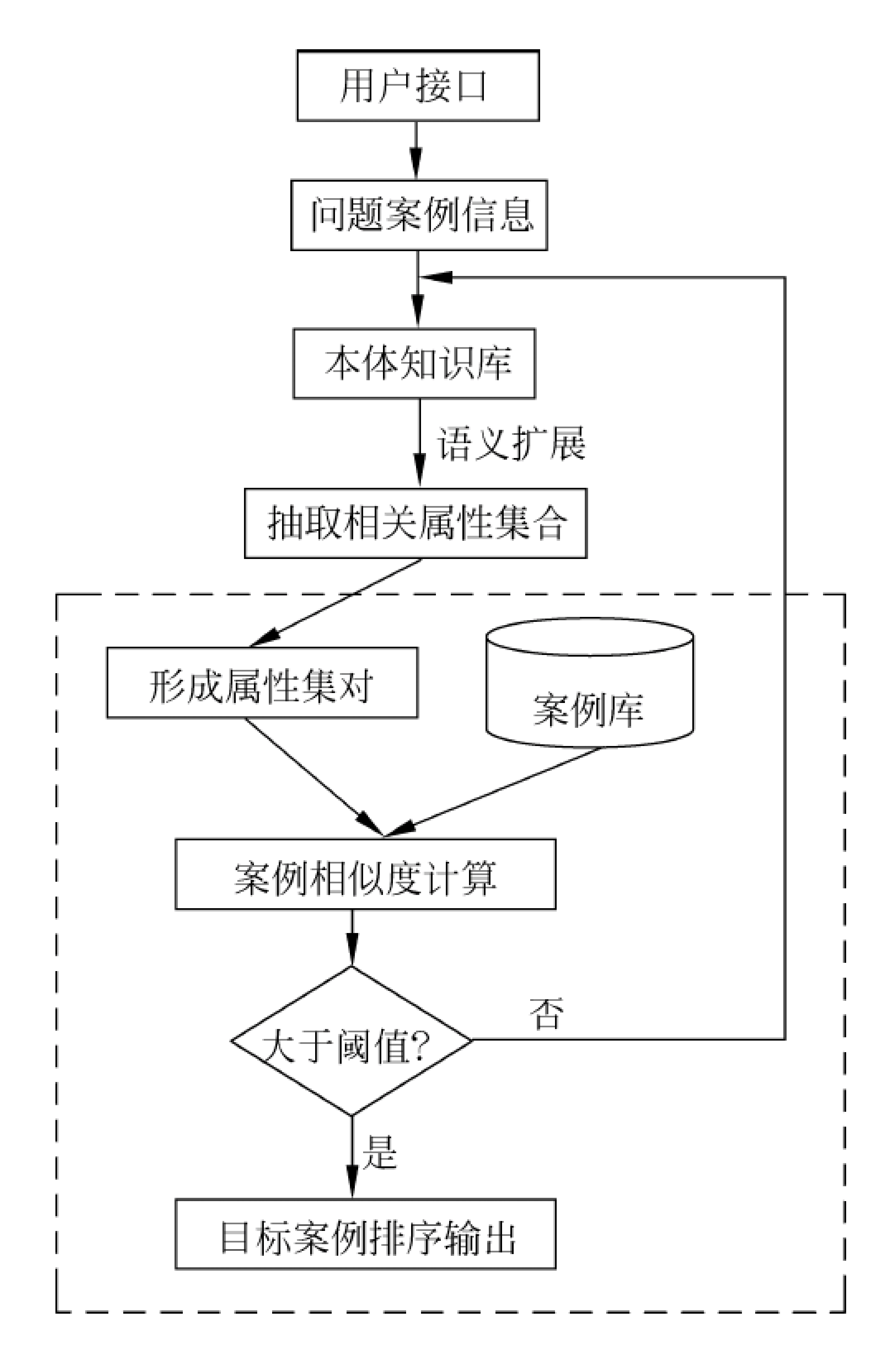


图 9-9 案例检索模型的系统原理图

第一步：用户输入问题案例 q 的描述信息并在问题背景下抽取关键词。

第二步：通过本体知识库进行近义或同义词的扩展，得出问题案例有关的属性集合，从而构造案例集对的属性集对。

第三步：对案例库中的每一个案例 p 应用公式(9-2)和公式(9-3)计算案例集对(q,p)的相似度，如果相似度大于设定阈值，将其存入目标案例库。

第四步：如果目标案例库为空，可以应用本体推理进一步扩展问题背景的属性集合，转第二步；否则转第五步。

第五步：按照得出的相似度值从大到小的原则重新将目标案例库排序。

9.3.4　中医诊疗的应用

通过人工收集和整理《丁甘人医案・伤寒案》、《中华医典》中所收录的名医案中涉及到的喘证案例，选取 600 余例喘证医案为研究对象，在领域专家的参与下并参考中医学相关标准，在排除概念的冗余性、歧义性及保证概念术语正确的前提下，选择“喘主证”、“喘息”、“喘逆”、“喘鸣”、“咳喘”及“上气”等关键词作为研究核心概念，筛选整理医案，最终确定相关属性字段十余个，建立了较为完善的中医喘证医案数据库，如图 9-10 所示，将此作为研究对象的形式背景。

	A	B	C	D	E	F	G
1	编号	喘主症	喘轻重	**痰象**	伴随症状（寒热、汗出、口渴、	舌象	脉象
2	目标	咳喘	中	痰白滑	纳呆，便秘	苔白腻	滑
3	1	喘	重	痰黄	状热	苔黄腻	滑数
4	2	咳喘	中	痰白滑		苔白腻	滑
5	3	喘息	重	痰白滑	自汗	舌淡，苔薄白	细软，滑
6	4	气短、喘	中		小便短赤，便秘	苔白	沉实
7	5	气短、喘	轻		小便少，泄泻	舌红，苔白腻	沉弱
8	6	喘促	中		微热	舌淡，苔白微腻	浮数
9	7	喘促	中		状热	舌淡苔腻微灰	沉数
10	8	微喘	中		状热，口渴	舌质红，苔微黄	浮数
11	9	喘促	重	痰黄	状热，泄泻，纳呆，便秘	舌红，无苔	虚
12	66	喘促	重	痰黄	微汗，纳呆，口渴，泄泻，小便少		
13	67	喘促	重	痰黄	状热	舌红少津，苔薄白而	沉数浮大，指
14	68	喘挣	中		壮热，纳呆		
15	69	喘			壮热，口渴，便秘，小便短赤，	苔黄燥	沉数，指纹粗
16	71	喘	重		壮热，口渴，小便短赤，纳呆	舌红，苔燥	浮弦，数
17	72	喘	重	痰清稀	纳呆，口渴，小便清长	苔白滑而腻	弦滑，弱，细
18	75	喘咳	中	痰清稀	纳呆，便秘，小便短赤，口渴	苔白腻	沉迟
19	76	喘咳	中	痰清稀	纳呆，小便清长	舌青色，苔白滑厚腻	弦滑
20	78	气短	中	痰少而黏	潮热，恶寒，自汗，盗汗，	苔白腻	虚数
21	81	喘挣	重		壮热	苔白滑	紧，指纹青黑
22	84	喘满短气	中	痰白	自汗，恶寒，口渴		

图 9-10　喘症医案数据库

1. 本体构建

在中医专家的参与下，对抽取的重要概念自顶至下定义出基本的类和层次关系，并用建模工具 Protégé4.1.0 建立中医喘证领域本体类关系的初始模型。其中，定义的类与本体结构中的类一致，对象与本体结构中的实例一致，例如，外感型喘是实喘的子类，而实喘又是喘证的子类，用本体语言描述这种关系如下：

```
<owl: Class rdf:ID="外感型喘">
<rdfs:subClassOf>
<owl:Class rdf:ID="实喘"/>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="实喘">
<rdfs:subClassOf>
<owl:Class rdf:about="喘证"/>
</rdfs:subClassOf >
</owl:Class >
```

定义各类之间的关系，每一种关系都是概念与概念间的一种映射，可以看成是二元组或多元组函数。函数定义域和值域的取值就是喘证领域内定义的类和子类的对象，每种关系相对应的 ObjectProperty 属性的 domain 子属性可以用来设置函数的定义域，关系对应的 ObjectProperty 属性的 rang 子属性可以用来设置函数的值域，如此可以使本体体系中各类的关联更加简单。以下显示的是 ObjectProperty 属性“痰象”以及它的特殊性质，其中“痰象”的定义域是案例，值域是由“痰白”、“痰黄”和“痰清稀”组成的并集构成，可以看出值域的并集构成了值域，而且值域还有传递属性。

(<owl：TransitiveProperty>)

```
<owl:TransitiveProperty rdf:about=" #痰象"
<rdfs: domain rdf: resource="#案例"/>
<owl: inverseOf rdf: resource="#构成"/>
<rdfs: range>
<owl: Class>
<owl: unionOf rdf: parseType="Collection">
<owl: Class rdf: about="#痰白"/>
<owl: Class rdf: about="#痰黄"/>
<owl: Class rdf: about="#痰清稀"/>
</owl: unionOf>
</owl: Class>
</rdfs: range>
<rdf: type rdf: resource=http://www.w3.org/2002/07/owl#ObjectProperty/>
</owl: TransitiveProperty>
```

建立的中医喘证本体片段如图 9-11 所示。

图 9-11　中医喘证本体片段

2. 数据的整理和相似度计算

表 9-2 所示数据为从案例库中选取的 12 个比较有代表性的案例,其中 2 个作为问题案例,并且使用中医专家推荐并结合本体结构的属性权重值。

表 9-2　中医喘证案例及属性权重值

案例	喘症	痰象(颜色)	寒热	舌体(色质)	舌苔(颜色)	舌苔(厚薄)	脉象
1	喘促	痰黄	状热	—	苔黄	薄	滑数
2	喘	—	微热	舌淡	苔白	—	浮数
3	喘息	痰黄	微热	—	苔红	略厚	弦滑
4	气喘	无	恶寒	舌淡	—	厚	浮数
5	气短	—	—	舌红	苔白	—	沉弱
6	喘嗽	痰黄	潮热	舌黄	—	薄	浮紧
7	咳喘	痰黄	—	舌淡	苔白	—	细滑
8	喘促	痰白	—	舌淡青	苔白	薄	细滑
9	微喘	痰白	恶寒	舌红	苔红	—	沉实
10	喘	痰白	微热	舌淡	苔白	薄	—
问题案例 1	咳喘	痰黄	状热	—	苔白	薄	滑
问题案例 2	气短	痰白	恶寒	舌黄	苔红	厚	浮
属性权重	0.4	0.15	0.05	0.1	0.05	0.1	0.15

为了验证本文提出的案例间相似度度量方案的高效性，尤其是某些属性的信息缺失或不确定情况下对案例间相似度度量的影响，分别对问题案例 1 和问题案例 2，采用一般相似度（海明距离法）计算和基于集对分析的相似度计算，阈值设定为 $\lambda=0.8$。计算结果对照数据见表 9-3。

3. 结果分析

实验结果如表 9-3 所示，可以看出，基于集对分析的相似度计算方法区分度更好，对于问题案例 1，用集对分析方法计算出大于阈值 $\lambda=0.8$ 的案例有 4 个，用海明距离法有 3 个，通过观察表 1 的数据且结合医案的诊断结果，发现案例 7 与问题案例 1 相接近，说明本文计算出的结果更接近事实。且用集对分析计算问题案例与源案例的相似度值比与源案例 9 的值大，但是用海明距离法结果却相反，可能是案例 5 属性缺失较多的原因，而集对分析法能有效的处理这些不确定信息。

表 9-3　实验结果

源案例	问题案例 1		问题案例 2	
序号	集对分析	海明距离	集对分析	海明距离
1	0.88	0.85	0.25	0.34
2	0.63	0.59	0.62	0.78
3	0.66	0.64	0.7	0.82
4	0.52	0.48	0.69	0.72
5	0.62	0.41	0.46	0.74
6	0.86	0.84	0.68	0.58
7	0.84	0.76	0.6	0.70
8	0.82	0.81	0.5	0.52
9	0.51	0.52	0.46	0.58
10	0.63	0.65	0.42	0.44

对于问题案例 2，用集对分析法计算出大于阈值的案例有 0 个而海明距离法有 1 个，通过观察表 9-2 的数据且结合医案的诊断结果，很明显，问题案例 2 与源案例属性不太接近，如果以海明距离法检索出的案例作为问题案例

求解的依据的话，诊断可能会缺乏准确性。由此我们可以总结出相对于基于距离的方法，本文对案例相似度的计算方法效果更好。

9.4　基于知识库的知识发现及应用

基于知识库的知识发现 KDK（Knowledge Discovery in Knowledge base），目的在真实的大型知识库中发现新的知识。这种发现过程借用 KDD 的语言说是“非平凡”的。意即这种发现过程的核心将是归纳，而演绎将作为辅助手段，该过程不同于传统的演绎，它有可能是不保真的。其次，KDK 能够发现深层次的知识。具体而言就是在已有关系的基础上进一步发现其上的关系，从逻辑角度上说就是发现谓词间的关系或涵词间的关系。再次，由于知识本身所可能具有的一些属性，如不确定性，非单调性，不完全性等，KDK 过程的进行也将是一个复杂的多方法多途径的过程。它与知识库的组织，用户对最终寻求的知识的类型都紧密相关，采用的推理手段可能涉及很多不同的逻辑领域。最后，KDK 发现的知识应该是有效的，潜在有用的，用户可理解的。这与 KDD 的要求相同。

KDK 究其本质来说应该是一种机器学习过程，其本质目的是获取知识，学习源是知识库，学习手段是用归纳结合演绎的方法，其最终结果将既能够发现事实上的知识，也能发现关系上的知识。KDK 要依赖于基于数据库的知识发现 KDD，因而能够很好的解决 KDD 的应用领域。

KDK 的潜在应用是十分广阔的，已经远远超出了最初的“货架子工程”。从工业到农业，从天文到地理，从预测预报到决策支持，KDK 都发挥着越来越重要的作用。

1. 商业方面

“货架子工程”是 KDD 最初成功应用的典范。也正是因为在商业方面的成功应用不断刺激着 KDD 的发展，进而拓展到越来越广阔的应用领域。如今商业，特别是销售业和服务行业，随着大数据时代的到来，已然是 KDK

应用最广泛的领域之一。主要应用于销售预测、库存需求、零售点选择、价格分析和销售模式分析。

2. 农业方面

农业是一个大型复杂系统，中国农业部门数十年来积累了大量的关于土肥、气象、病虫害、市场信息等方面的数据、实例和经验知识。但基本上没有得到充分利用。如果将这些知识通过本体构建知识库，不仅实现知识共享，还可以从中发现许多有价值和有规律的隐含知识。如通过对病虫害数据的分析，可以发现病虫害的影响因素、迁移或蔓延规律等，从而遏制灾害的发生、扩展或降低灾害损失，通过对国际国内市场信息的挖掘来指导农业生产规划等。

3. 医学生物方面

医疗保健行业有大量数据需要处理，但这个行业的数据由不同的信息系统管理，数据组织性差，而且类型复杂。如医疗诊断数据，可能包括文本、数值，图像等，都给应用带来了一些困难，统一的资源描述和知识表示成为应用的瓶颈。KDK 在医药方面主要用于医疗诊断分析、药物成分一效用分析、新药研制和药物生产工艺控制优化等。

4. 金融保险方面

金融事务需要收集和处理大量数据，对这些数据进行分析，发现其数据模式及特征。然后可能发现某个客户、消费群体或组织的金融和商业兴趣，并可观察金融市场的变化趋势。KDK 在金融领域应用广泛，如金融、股票市场分析和预测，账户分类，银行担保和信用评估等。

5. 通信、媒体方面

如线路故障的预测、收视率的影响因素、网站入侵检测、Web 信息发现等。

6. 国防军事方面

如军事情报资料分析、指挥自动化与辅助决策、战争风险预测、武器攻击效果分析、地理数据分析等。

7. 其他方面

如工业生产中设备故障诊断、生产工艺优化；科学研究中的数据处理与分析、气象分析和预报等。

在所得到的大量信息中，包括有结构化的数据，半结构化的数据（文本信息等）和非结构化数据（图形、图像、视频、音频信息等）。而 KDD 技术只处理其中以结构化的数据类型。与变化不定的数据比较，知识是相对稳定的、是从较高的层次看数据、是数据的抽象表示，其表达的内涵要远远大于数据信息。因此，研究知识中内在的联系，将成为知识发现领域下一步的研究目标。正像 KDD 技术的产生是由于数据爆炸而知识缺乏一样，随着知识信息的不断增加，知识爆炸能提供给决策者有效的决策信息严重缺乏的情况日益引起人们的注意，KDK 将成为知识发现的研究热点和应用技术。

国际上目前针对语义推理的热点研究领域主要涵盖了以下几个方面：

（1）理论方面

由于 OWL 的逻辑基础是描述逻辑，因此扩展描述逻辑表达能力并开发对应的正确有效的推理算法成为了人们研究的热点。

（2）功能方面

支持规则推理，如支持 SWRL 便是典型的一例，这样的系统有 Bossam；支持多版本本体查询与推理，典型的项目有 MORE；支持不一致本体的推理，典型代表有 PION；支持分布式本体的推理，较新的项目有 DRAGO；另外优化用户界面以及为开发用户提供更为丰富的程序开发接口等都是本体推理机发展的最新趋势。

（3）应用方面

进一步为语义 WEB 七层模型中的证据层提供解释服务，这样的参考应

用系统有美国斯坦福大学的 wineAgent 项目；进一步结合本体编辑器，为用户提供 Debug 服务；另外把本体推理机更好地集成到大型本体服务器中都是本体推理机应用的最新趋势。

参考文献

[1] 张佩云，孙亚民，吴江. 基于本体的知识检索研究及实现. 情报学报[J]，2006(10)，553-558

[2] 阎红灿，王会芳，马会霞. 基于集对分析的案例检索模型. 微型机与应用[J]，2014(21)

[3] 马森，赵文，袁崇义等. 基于规则推理的语义检索若干关键技术研究. 电子学报[J]，2013(5)

[4] Yan Hongcan，Wang Huifang，Zhang Shufen，et al. Ontology case retrieval based on SPA and asthma diagnosis application [J]. Journal of Chemical and Pharmaceutical Research，2013(12)：587-593

[5] M. Andea Rodriguze，Max J. Egenhofer. Determining Semantic Similarity among Entity Class from Differernt Ontologies[J]. IEEE Transations on Knowledge and Data Engineering，2003，15(2)：442-456

[6] 阎红灿，闫宏图，刘保相. Tableau 算法在粗逻辑知识推理中的应用，贵州师范大学学报(自然科学版)，2013. 1(31)：40-43.

[7] 齐红，张亮亮，李昕等. 基于玉米本体的语义检索系统，计算机工程[J]，2011(2)，34-37

[8] 纪兆辉. 本体的推理研究[J]. 南京师范大学学报(工程技术版). 2012，12(9)：54-59

[9] 张艳涛，陈俊杰，相洁. 基于 SWRL 本体推理研究[J]. 微计算机信息. 2010，26(3)：182-184

[10] 马森，赵文，袁崇义等. 基于规则推理的语义检索若干关键技术研究[J]. 电子学报，2014，5(41)：978-981

[11] 唐晓波，金钟鸣. 基于本体与规则的语义推理研究[J]. 情报学报，2013. 7(30)：695-703